Enhancing Multi-Robot Coordinated Teams with Sliding Autonomy

Jonathan D. Brookshire

TR-04-40

The Robotics Institute
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15232

July 2004

Submitted in partial fulfillment of the requirements for the degree of Master of Science

Table of Contents

Chapter 1 Introduction

The combination of human intelligence, autonomous precision, and robotic strength has the potential to bridge the gap which leaves many tasks especially suited to robotics beyond the reach of existing technology. The beginnings of this human-robot amalgamation are already evident in the Spirit and Opportunity missions to Mars. With sizable delays and narrow windows of communication, a system that can work autonomously on the Martian surface and integrate new periodic requests from Earth is essential. This mixture allows the robot to perform straightforward but lengthy tasks, such as grinding a rock surface or traveling from point A to point B, during communications blackouts. Should the autonomous system encounter some unanticipated fault, perhaps a stalled grinding wheel or a steep cliff, the system can halt and await instructions from Earth. The remote human operators can then plan appropriate recovery strategies and issue new commands. Limited time, energy, bandwidth, and access to human operators prohibits a purely teleoperated approach. Inadequate sensing and artificial intelligence combined with changing mission goals and unexpected contingencies make a purely autonomous system impractical. The goal is to mix human and robotic talents, through what we term sliding autonomy, to achieve the highest possible overall performance.

With sliding autonomy, both the human operators and autonomous agents can be assigned control over different parts of a single system. These control assignments can also be changed, allowing the operator or the autonomous system to request help from the other. Moving beyond single robot systems, then, this thesis explores the use of sliding autonomy with heterogeneous, coordinated robot teams. The integration of multiple robots will be essential to leverage the unique skills of each robot to accomplish tasks that no single robot could accomplish alone. Simultaneously, human input will increase system performance by improving the exception handling capability, simplifying autonomous operation, and boosting speed and reliability.

1.1 Motivation

Sliding autonomy has the potential to affect many aspects of advanced robot use. The Space Solar Power (SSP) project, part of which funds Trestle research via NSF, NASA, and EPRI, investigates the possibilities of constructing a large solar electricity generation facility orbiting the Earth. A large scale solar resource could provide a cheap, clean, and reliable source of energy. Above the Earth’s atmosphere and weather, the Sun’s rays could be collected more efficiently and the energy beamed (possibly with high-energy microwaves) back to the surface.

At the time of this thesis a design for the SSP is not finalized, but several initial estimates suggest the facility will be kilometers in size and require several tons of material (see Figure 1). When operational, the installation will require maintenance in a high voltage and high temperature environment. Construction directly by astronauts would require months of tedious assembly in a dangerous environment. A purely teleoperated robotic system would alleviate the danger to humans, but would come only with an increased workload for ground operators and delay production. On other hand, the construction and maintenance tasks are far too complex for a completely autonomous robotic system. The solution, it seems, is to employ sliding autonomy.

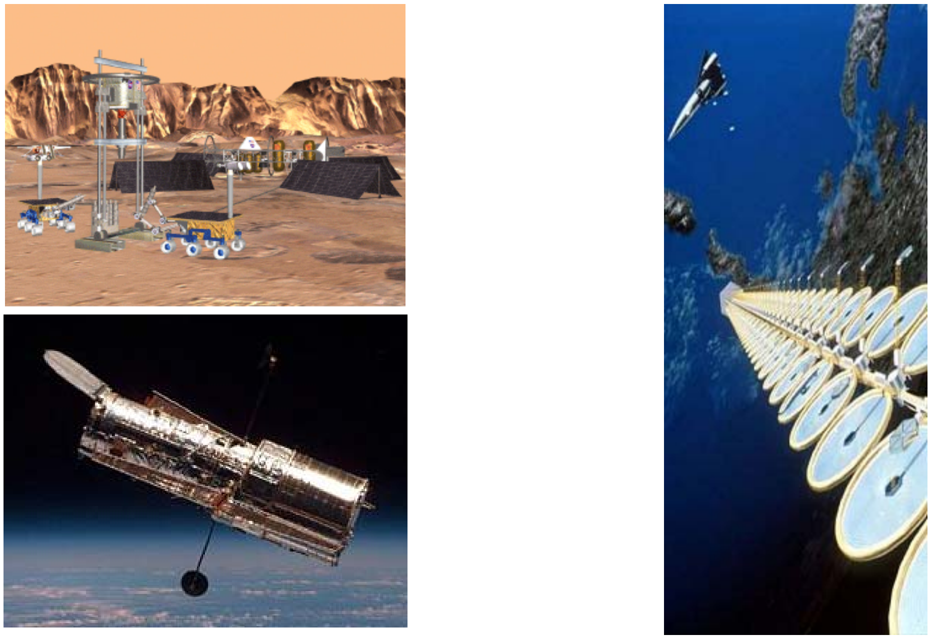


Figure 1 – Planetary habitat construction (upper left), Hubble telescope repair (lower left), and Space Solar Power plant construction (right) are a few of the tasks which exist that would benefit, and even require, a robot workforce. Unfortunately, many of the tasks are too complex for even the latest autonomous systems. By employing a mixture of human and autonomous control, a team of robots can handle a much wider variety of tasks and situations.

For similar reasons, a system capable of sliding autonomy would also be appropriate for moon or planetary construction. Habitats may need to be constructed before humans arrive, presenting the same kinds of complex assembly challenges as the SSP. Furthermore, the robotic workforce may evolve as new robots arrive, replacing old units and adding different capabilities. The result will be a heterogeneous robot team the task of which is to assemble complex structures in a hostile, and often unknown, environment. Such a system requires not only sliding autonomy, but also coordination between a large team of robots.

Even more immediate applications could benefit from a coordinated team of robots and humans. The recent Columbia shuttle disaster has raised safety concerns and restricted shuttle focus to the completion of the International Space Station. NASA reports that, without regular servicing from shuttle astronaut crews, the Hubble space telescope will last only a small portion of its potential life. NASA contends that the technology to robotically repair the satellite would have to be developed “lickety-split” and is beyond the state-of-the-art (Cass 2004). Although it is unclear if a team of robots could be developed and deployed in time, it is clear that such a system would benefit from tight human-robot coordination.

1.2 Coordinated Teams

Much of the work done researching the coordination of multi-robot teams was inherited from the Distributed Robot Architecture (DiRA) project. The DiRA project developed and tested a software architecture that enabled a group of robots to be loosely coupled yet capable of tight integration.

Each specialized robot plays a role in docking a beam securely between two uprights (see Figure 2). This task is representative of a task that might be needed in space construction. A large crane provides the heavy lifting capability and large workspace to grossly maneuver the beam, while a smaller mobile manipulator finely positions the beam into the docking clamps using a coordinated resolved motion rate control to drive the end-effector. A mobile vision system provides feedback for visual servoing and can be moved to focus on different aspects of the operation. The beam and the robots are marked with fiducials that allow the vision system to determine a relative pose between the fiducials. This information is then continually transmitted wirelessly to the two other robots so that they can, in turn, use this information to move.

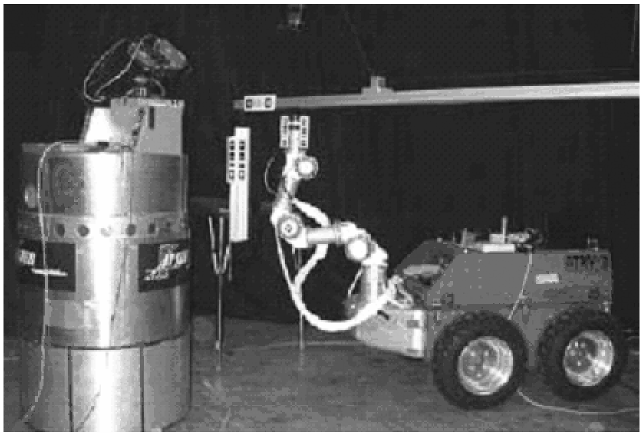


Figure 2 – A crane (not visible), mobile manipulator (right), and mobile vision system (left) work together to dock a horizontal beam into two vertical receptacles (second receptacle not visible). The task provides a test-bed for examining multi-robot coordination in an assembly environment.

Each individual robot in the team is responsible for negotiating with other agents as a plan is developed, coordinating its portion of derived task tree, and managing its unique skill set to achieve tasks. The resulting system is decentralized since each robot manages its individual autonomy. Decentralization also makes for a more robust system since there is no single point of failure (a comparison example of centralized control can be found in Khatib 1995).

1.3 Sliding Autonomy

The goal of sliding autonomy is to dynamically merge the good qualities of both humans and robots into a single system. Typically, sliding autonomy, also known as adjustable autonomy, is described as the middle region between a purely teleoperated (or remote controlled) system and a fully autonomous system (see Figure 3).

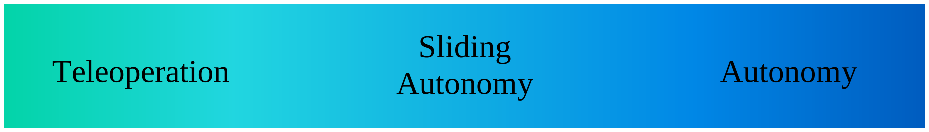


Figure 3 – Sliding autonomy region describes modes of system operation that combine elements of both human and autonomous control.

It is important to note that the "Sliding Autonomy" region in Figure 3 is not simply one system "mode," but rather a continuum of different system modes. Minimally, a system capable of sliding autonomy can be switched between the extreme teleoperated and autonomous modes during operation. This kind of switching allows for exclusively human or autonomous control.

We wish, however, to describe the range of sliding autonomy. Individual components within the system can be *switched* to create a discrete set of autonomous levels. Each level is a permutation of the switchable components in either autonomous or teleoperated mode. For example, on a planetary rover, the task responsible for preventing collisions may run autonomously while Earth-based operators attempt to navigate through treacherous terrain. Should the operators issue a command that would lead to a collision, the autonomous collision avoidance module would, at the very least, notify them of the danger. The switching of tasks can be initiated by the robot, scripted, or initiated by the human. When the robot reaches a state where it does not know how to proceed, it may initiate a mode switch and request that the operator handle the exception. Certain tasks may demand operator assistance, these tasks can be pre-scripted to be carried out with operator assistance. A human may also initiate control when the autonomous system requires additional guidance (but may not realize its need).

Sliding autonomy can also be introduced into the system in a variety of other ways. The operator may provide the system with parameters, suggesting, for example, that the planetary rover avoid some hazard by a greater distance. Operators can also be involved in dialog with the autonomous system, helping to resolve ambiguities. At more abstract levels, humans may alter system-wide goals (e.g., by specifying what areas to explore) or suggesting alterations to automatically generated plans (e.g., reordering exploration sites to save energy).

1.4 Thesis Goals

The goal of this thesis is to develop, implement, and evaluate sliding autonomy with a coordinated group of heterogeneous robots performing construction tasks. Specifically, this thesis will argue that a system that incorporates sliding autonomy will perform more reliably and with greater speed that both fully autonomous and teleoperated versions of similar systems. Moreover, this thesis will demonstrate an architecture within that of the coordinated robot team environment that supports sliding autonomy in a structured manner.

This thesis will begin assessing the hypothesis by developing and implementing a framework for sliding autonomy. Next, it will present a series of experiments comparing autonomous, teleoperated, and sliding autonomous versions of the DiRA assembly task. The results support the original assertion.

The discussion of sliding autonomy also formalizes several ideas that developed during the work. (1) Sliding autonomy can fulfill the responsibilities of otherwise autonomous exception handlers. When comparing autonomous and sliding autonomous systems, it is important that the sliding autonomy handle genuine exceptions rather than simply conceal deficiencies in the autonomous system. If, for example, the autonomous system is 100% unreliable, then sliding autonomy will doubtlessly make the system better. (2) In a similar fashion, the comparison of teleoperated and sliding autonomous systems can be biased. An advanced user interface may offer significant advantages over a less well developed interface. The discussion attempts to ascertain what constitutes purely teleoperated control. (3) The implementation of sliding autonomy focuses on creating a task-centric control. The architecture attempts to concentrate the operator's attention on monitoring and controlling tasks, rather than on individual robots. (4) A system capable of sliding autonomy must preserve state information. The system's task tree maintains this state and allows the autonomous system to resume gracefully after teleoperation. (5) During the research, sliding autonomy emerged as a useful debugging tool. Similarities between sliding autonomy and typical software debugging practices are presented.

1.5 Thesis Overview

Chapter 2 will discuss and compare related work from the wide range of sliding autonomy research. Chapter 3 will focus on the details of the Trestle system architecture including a discussion of the robots involved, the related assembly task, and details of the systems implementation. Chapter 4 will convey the specifics of the experiments performed and present the raw results. Chapter 5 investigates sliding autonomy from a system analysis approach using Petri nets. Chapter 6 will synthesize the experimental data, draw conclusions, and discuss extensions of the sliding autonomy architecture. Chapter 7 describes the next assembly task and test-bed for sliding autonomy validation. Appendix A is a primer for Petri Networks and Appendix B evaluates several planners for the Trestle assembly task. Appendix C discusses the details of a crane controller. Appendix D proposes a means to calibrate an alternative to visual servoing and Appendix E overviews principles of visual servoing.

Chapter 2 Background

We wish to design an architecture that is capable of mixing human and autonomous control strategies during execution. The combination of control will create systems capable of sliding autonomy. Synonymous with the commonly used phrase adjustable autonomy, sliding autonomy refers to the ability to dynamically vary the involvement of the human operator during the system's execution.

There are many dimensions to sliding autonomy, but this thesis focuses on task level switching. At the extremes, the system may be purely autonomous or purely under human control (teleoperation). That is, at these extremes, the primary task (e.g., "assemble the structure") is done entirely by the autonomous or by the teleoperated system. A scalable region of operator and autonomous interaction exists between these extremes. Suppose the goal to "assemble structure" decomposes into "grasp beam" and "dock beam" tasks (see Figure 4).

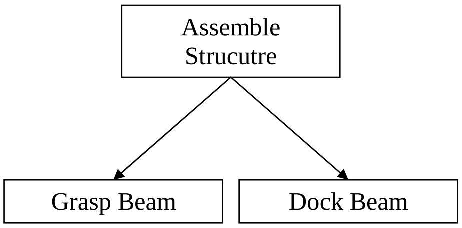


Figure 4 – Tasks are decomposed into smaller, simpler sub-tasks. Each "task unit" can be either done by the operator or by the autonomous system.

If all tasks are controlled by the operator, the system is in a purely teleoperated mode. If all tasks are controlled autonomously, the system is in a purely autonomous mode. If the operator executes "grasp beam" and the autonomous system performs the "dock beam," the system exhibits an intermediate level of sliding autonomy. The level of sliding autonomy may change during executing. After the operator competes the "grasp beam," he may wish to also switch control of the "dock beam" and perform it manually as well.

There are three ways that a task may change modes (i.e., whether it is performed autonomously or through teleoperation):

1. Preassigned switch – A task is determined a priori to be performed autonomously or by the operator. Tasks for which operator control is preferred, or no autonomous control is available, may be designated preassigned to the operator.
2. Human-initiated switch – During execution, the operator may wish to switch the mode of a task. The operator would intervene when the autonomous system is performing inadequately or has failed.
3. Robot-initiated switch – The autonomous agents may request operator assistance when unsure or unable to complete some task. Measures of uncertainty could be

a function of anything ranging from a probability analysis of the robot's state to a simple retry or timeout limit.

Let the set of tasks which can be competed by the operator be O and the set of tasks which can be competed by the autonomous system be A. All tasks than can be completed only by the operator, $O - O \bigcap A$, must be preassigned to teleoperated control. All tasks that can be competed only by the autonomous system, $A - O \bigcap A$, must be preassigned to autonomous control. Tasks capable of autonomous and teleoperated control, $O \bigcap A$, can be dynamically switched during execution between the two modes. All of the tasks, $O \bigcup A$, represent the system's total task capabilities.

2.1 Related Work

A wealth of work concerning human-robot and human-software agent interactions exists, but this chapter concentrates on the work most related to sliding autonomy (especially with multi-agent robotic teams). The term sliding autonomy is interchangeable with adjustable autonomy as presented by Dorais et al. (Dorais 1998). The authors provide several examples in which sliding autonomy will be essential for space operations where demands on the operator must be focused and minimized. During a studied 600 day stay on Mars, the authors evaluate the kinds of tasks human astronauts will perform. Whereas many of today's space missions receive continuous round-the-clock support, the crew will be heavily burdened and will need to rely on and interact with a variety of autonomous systems. Autonomous/human interactions will include mission planning, robotic exploration, and system maintenance. The authors conclude that the goal of sliding autonomy in these kinds of situations is to maximize the system's capabilities and minimize the human attention required. Sliding autonomy should also make the autonomous system more versatile and easier to understand.

The different aspects of adjustable autonomy are formalized by Barber et al. (Barber 2001). The authors segment adjustability into independence, control, and goal dimensions. The independence dimension allows the operator to vary the number of options an autonomous decision maker has. During a path planning search, for example, the decision maker's independence would be limited if the human marked certain regions or paths as impassable. The control dimension allows the operator to vary the ability of a decision maker to choose a particular option. In a path planning search, the operator might instruct the system to use a particular starting path segment. Control can be adjusted by modifying the relative weight each agent has in the decision making process (e.g., the human operator might make his own weight sufficient to subdue all autonomous agents). The goal dimension allows the operator to adjust the goals. In the path planning example, the operator would simply alter the planner's destination.

Goodrich et al. (Goodrich 2003) characterize sliding autonomy by robot effectiveness as a function of neglect. Effectiveness is a subjective measure of the robot's output (e.g., goals satisfied) and neglect is related to the time since the autonomous system was last serviced (e.g., related to communication latencies). A teleoperated system has a potential for very high effectiveness, but has little tolerance for neglect. A purely autonomous system is immune to neglect, but has a fixed effectiveness. Sliding autonomy is effective with little neglect, but its performance degrades gracefully as at the neglect increases. Thus, sliding autonomy achieves a teleoperation-like effectiveness with substantial operator attention and an autonomous-like performance with significant negligence. The authors explore neglect and effectiveness with a robot/human system tasked with remote exploration. The operator is encouraged to neglect the system while performing increasingly difficult mental math. A ratio of performance with user interaction to performance without user interaction has also been proposed as a measure of autonomy (Brainov 2001).

Using a roving eye and a fixed manipulator similar to Trestle's Mobile Manipulator (see Section 3.1.1), Kortenkamp et al. developed and tested a software infrastructure that allows for sliding autonomous control of a robot manipulator (Kortenkamp 1999, 2001, 2002). The system's task was to perform repairs on a circuit board. To perform the task, a shielding blanket had to be removed (a task that could only be done manually). Then, the lid and two securing bolts must be removed. A battery, two more bolts, and a set of crystals had to then be manipulated. The task was chosen because of its space relevance, the need for human manipulation, and opportunities for sliding autonomy. Note that, unlike the Trestle system, the human actually worked alongside the robot. The authors made use of a task decomposition similar to Trestle's. Each task was assigned a mode and the operator could change that mode during runtime. All sub-tasks of a teleoperated task were also teleoperated. When a task mode changed from operator to autonomous control and the state of the world was ambiguous, the system either performed a visual scan of the world to determine the positions of all objects or selectively searched for important objects. The operator could also interact with a high-level planner and make changes to the plan during execution. Of the surveyed literature, this work most closely resembles the Trestle project. The research associated with this thesis extends this work by taking a different architecture focus (primarily the task level operation) and attempting to identify in what way sliding autonomy can be most beneficial.

Several papers from NASA Langley (Herstrom 1992, Rhodes 1994, Doggett 1996) describe a robotic system that constructs a large 102-strut, eight meter tetrahedral truss structure (see Figure 5). The work provides close detail at a mechanical and software level. Several base nodes are fixed to a rotary platform that serves as the foundation for the entire structure. A single robot manipulator retrieves beams from a hopper and assembles them, relying on a visual servoing system. Although the primary focus of the technical reports is on the assembly procedure, the software describes a system by which a beam's placement can be paused and even reversed during execution. The control software describes the relationship between installing a beam and reversing a beam installation and how the two are not necessarily simply a reverse list of the same actions. When paused, the operator could resume, move the end-effector and resume from the new point, or reverse the end-effector back to its starting position.

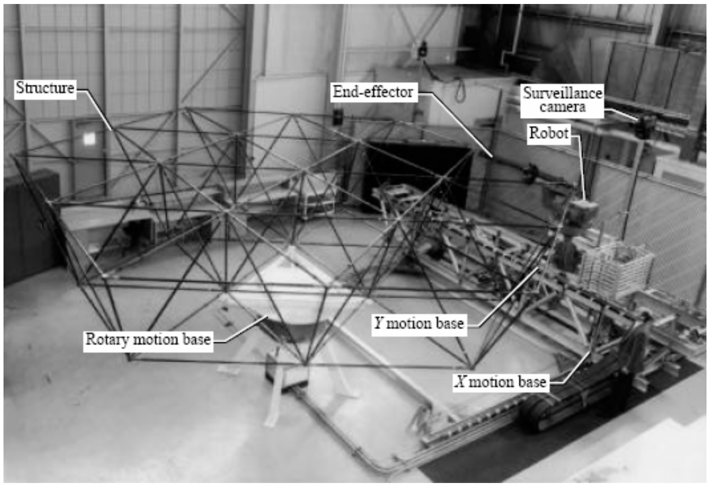


Figure 5 – An eight meter in diameter truss structure constructed at NASA Langley allows the operator play, pause, and resume functionality. (Rhodes 1994)

Stein and Paul investigate sliding autonomy as a way to increase productivity in environments with substantial communication delays (Stein 1993). With varying latencies, data is collected from users who are teleoperating a robotic arm to cut thermal blanketing (a task that might be required for satellite repair). The authors argue that the user can maintain the system with a mixture of human controllable low-level behaviors (e.g., the cutting operation) and higher level task monitoring capabilities. During communication blackouts, the autonomous system can still make progress toward the goal.

Sliding autonomy can also benefit what might be traditionally considered automated systems. An automated system, such as the Water Recovery System (WRS) system described in (Schreckenghost 2004 and Martin 2004), was used by NASA to study maintenance and operation over a period of 16 months. The WRS was responsible for purifying waster water by removing organic and inorganic materials. The system operated virtually human-free (typical of other life support systems). Operators must, however, perform routine maintenance on the WRS which cannot simply be turned off (a complete shutdown would be timely, costly, and might kill biological agents necessary for proper operation). The result was that the operators and autonomous system had to coordinate maintenance activities. The authors proposed and tested a system that allowed operators to schedule their work based on the requirements of others and the state of the automated system. A related system (Martin 2003) describes liaison agents which interface to autonomous agents on behalf of humans.

Miller et al. describe a human/robot system for control of an Unmanned Combat Air Vehicle (UCAV). The authors propose that in order for a human to develop an intuitive robot plan, the autonomous planner and the human must use the same vocabulary and have the same context. For example, when planning a UCAV combat profile, the human should expect the system to have the context of a typical pilot. As planning begins, the operator should be presented with a skeleton task tree (not necessarily complete in the AI planning sense) which the operator can modify and supply with additional information. The system should prompt the user for additional necessary information (e.g., reconnaissance target) in the same way a pilot would request the information before flying a mission. Ideally, operators could also start unique plans from scratch by piecing together primitives (e.g., fly to waypoint). Once the UCAV is in flight, the plan can be updated as more information becomes available by specifying the way sub-tasks are decomposed.

Underground mining has also seen a recent trend away from direct human control toward semi-autonomous mining vehicles. Although teleoperation removes miners from hazardous environments, it still requires the full attention of a dedicated operator (Ward 2003). Increasing levels of autonomy allow a single operator to control multiple mining machines more reliability (damage to equipment is commonplace in narrow corridors). Mining machinery is typically enhanced with autonomous drive capabilities. The system autonomously drives Load Haul Dump vehicles between locations using proximity tags located in along navigation routes. The operator is only employed during sensitive dumping and loading procedures (Steele 2001).

The COBOT project seeks to make manually operated machines more intelligent by providing guidance so that the operator does not have to finesse control. Typically, the human provides the force input, while the system steers the mechanism into the right place (Gillespie 1999 and Wannasuphoprasit 1998). Another system is described in which the robot and the user participate in a dialogue (Fong 2001, 2003). The robot can ask the operator to help with localization or to clarify sensor readings. The operator can also make queries of the robot. This framework assumes that the robot is capable of performing all tasks as long as it has full state information. A probability analysis about when to prompt the user with questions can also be developed (Fleming 2001).

Several examples of architectures for sliding autonomy motivated with a need for daily schedulers exist (Scerri 2002, Berry 2004, and Sierhuis 2003). The autonomous systems attempted to resolve timing conflicts (missed meetings, group discussions, personal conflicts, etc.) among some set of team members. Members could adjust the autonomy of the system by indicating their intent to attend gatherings or willingness to perform tasks. Analogous work with scheduling and resource satisfaction has also been performed (Corkill 2003).

A body of work also exists that characterizes human-computer and human-robot interactions. Although related, the work tends to describe systems capable of exclusive teleoperation or autonomous control and its impact to this work is minimal. The work presented here creates the foundation for sliding autonomy on which this thesis is built.

Chapter 3 The Trestle Project

The Trestle project seeks to build on the technology developed in the Distributed Robot Architecture (DiRA) project. The goals of the DiRA project were to investigate and develop the abilities of a group of multiple heterogeneous robots to function as a team. In particular, a robot team where each robot manages its individual capabilities and goals, and works towards those goals independently, was designed. Enabling this independence was a software architecture that loosely linked agents and allowed tight coordination (Simmons 2002).

This chapter will begin by describing the elements of the DiRA and Trestle projects. The implementations of sliding autonomy unique to the Trestle project will then be discussed.

3.1 The System

A practical example of tightly coupled, yet independent, agents can be found at a typical construction site. Shown in the Figure 6 is a construction site where a crane is lowering part of a building's steel support structure into position.

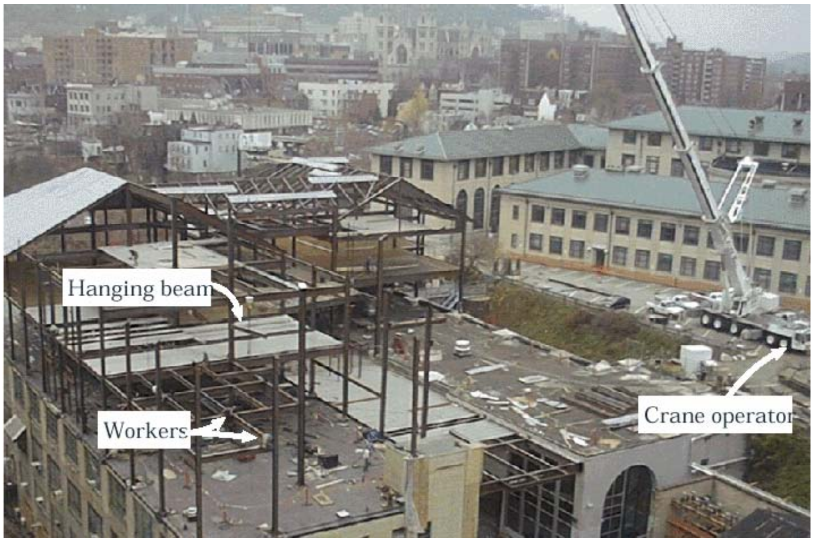


Figure 6 – The construction of a building requires the coordination of and communication between many independent workers.

There are several agents working together to bring the steel beam into position. The crane's operator performs the gross manipulation of the beam. Several workers align the beam precisely into its final position. Simultaneously, other workers provide the crane operator with visual feedback helping to clarify the crane operator's otherwise obscured view. For this kind of large scale assembly, it is clear that none of these agents could complete the assembly task alone. There is also no central coordination between these agents. A central site foreman may have assigned roles, but afterwards each agent is responsible for coordinating himself with other team members and for satisfying his own job goals.

This project explores the concepts of multi-agent coordination in the context of an assembly task not unlike that of the construction site. The system's goal is to dock a large beam into a pair of receptacles, as shown in Figure 7. The beam is approximately three meters long and the tolerance of the docking clamps is about five millimeters. The resulting task is difficult and requires considerable coordination between a team of robots. The robot team is comprised of a crane, a mobile manipulator, and a roving eye.

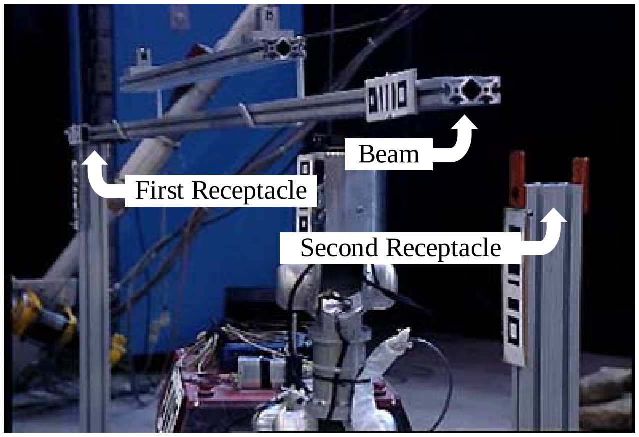


Figure 7 – The task is to dock the horizontal beam between the first and second vertical receptacles. The beam is several meters long and the tolerance of the docking receptacles is only a few millimeters. No single robot of the three-robot team is capable of docking the beam alone and, so, they must cooperate.

3.1.1 The Robots

The NIST Robocrane, or Crane, provides the heavy-lifting and gross manipulation capabilities of the system. The Crane is a 6-DOF Inverted Stewart Platform with a workspace of several meters. The triangular end-effector is one meter on each side and is suspended from the Crane's triangular base structure which is about six meters on each side. Actuated cables connect each vertex of the base triangle to two vertices on the end-effector for a total of six joints. The Crane can be controlled in absolute and relative position mode or velocity mode (see Appendix C for details). The beam is suspended from the Crane's end-effector with a simple pulley system which gives the beam extra compliance needed during docking.

Unfortunately, the Crane lacks the repeatability and accuracy to reliably dock the beam between the receptacles. The Mobile Manipulator, on the other hand, has sufficiently high resolution for docking but lacks the strength to support the beam alone. To accomplish the assembly, the Crane and Mobile Manipulator must work collectively. The Mobile Manipulator is a hybrid composed of a skid-steered RWI mobile base and an attached 5-DOF arm developed by NASA. The base and arm are coordinated through a CMU developed Resolved Motion Rate Controller (RMRC). The arm has intrinsic workspace limitations (as do all such manipulators). When mounted to the mobile base, the workspace is greatly enlarged and the end-effector can be positioned with a multitude of arm/base configurations. RMRC determines the configuration most appropriate by placing the arm in a position of high manipulability (distance from singularities) and moving the base appropriately (Shin 2003). RMRC is capable of position and velocity based end-effector control.

The Crane and Mobile Manipulator have only rudimentary sensing capabilities (i.e., limit and contact switches). In order to interact successfully with their environments, the Roving Eye provides positional information. The Roving Eye is constructed on top of a 24 inch diameter, four-wheeled, synchro-drive base, built by RWI. Sensors include bump panels, a Denning sonar ring, a Nomadics laser light striper, and a black and white stereo camera pair mounted on Directed Perception pan/tilt head. The sonar ring and light striper are not used for this research. Instead, the stereo pair provides the 6-DOF poses of fiducials placed on key objects in the environment. The fiducials are barcode-like schemes from which the Roving Eye can determine a 6-DOF pose relative to the camera. The fiducials, as show in the figure below, also convey information identifying what they represent (e.g., a particular fiducial represents itself as being on the beam).

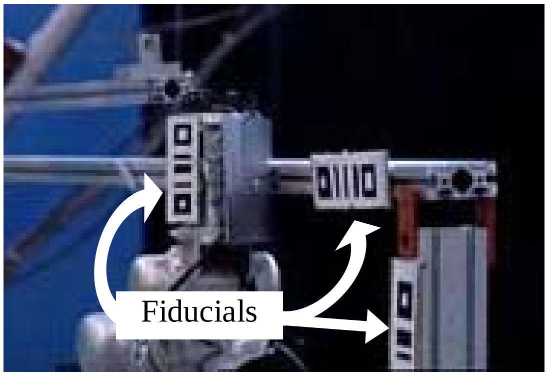


Figure 8 – Fiducials are used by the Roving Eye to determine the pose of key objects during assembly. The Roving Eye determines the pose of a fiducial relative to other fiducials and, as a result, does not depend on the position of the Roving Eye. Here, the system is servoing the end of the horizontal beam relative to the vertical stanchion.

The Roving Eye is, as the name suggests, able to move and position the cameras so as to achieve the best view of the interesting fiducials. A fiducial is of interest if the Crane or Mobile Manipulator has requested the pose of the fiducial's object. The Roving Eye provides the poses of fiducials relative to each other. Since the poses of fiducials are calculated relative to the camera, relative fiducial poses can be provided without knowing the position of the Roving Eye itself. As a result, accumulated errors in the position of the Roving Eye do not affect the accuracy of the fiducial poses. The Crane and Mobile Manipulator then use this pose information as feedback in their visual servoing loop. For more information on visual servoing, see Appendix E.

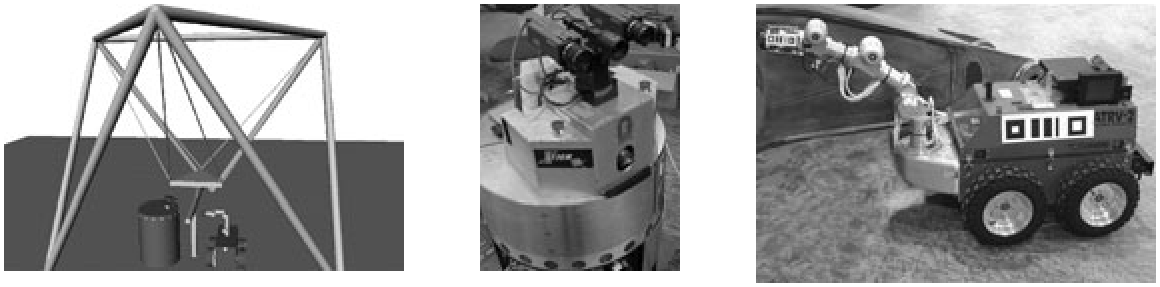


Figure 9 –Experimental project test bed consisting of a (left) six DOF crane, (middle) roving eye, and (right) mobile manipulator. The Crane is responsible for heavy lifting and gross manipulation. The Roving Eye is responsible for providing visual feedback for servoing. The Mobile Manipulator provides the fine manipulation for docking the beam.

3.1.2 The Task

The task of the robots is to dock the horizontal beam between the two vertical receptacles (stanchions) (see Figure 7). The docking procedure in Figure 10 consists of three primary steps: docking the first end ("near end dock"), relocating to the second dock site ("swap ends"), and docking the second end ("far end dock").

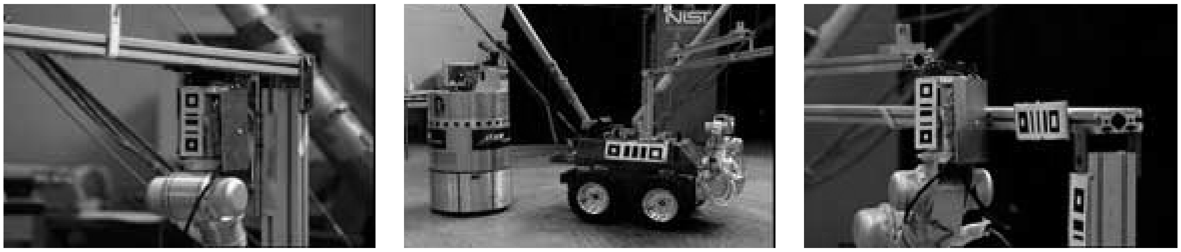


Figure 10 – Steps in the assembly process. Crane brings the beam close to the upright supports. The Mobile Manipulator grasps one end and docks it in one support. ("near end", left) The Mobile Manipulator turns around, drives to the second support. ("swap ends", middle) In this case, the Mobile Manipulator guides the beam in the horizontal direction while the Crane lowers the beam into place. ("far end", right)

During the near end, the Crane brings the beam into a rough position above the first stanchion. From this point, there is sufficient compliance (since the beam is actually suspended from the Crane) that the Mobile Manipulator can now energize its electromagnet and grab the beam. The Mobile Manipulator will guide the beam through a series of waypoints as it completes the first dock. Next, the Mobile Manipulator will release the beam, stow its arm and turn 180°. Because part of the beam is now fixed (docked in the first stanchion), there is no longer sufficient compliance for the Mobile Manipulator to simply grab and dock the beam. For the second end, the degrees of freedom are separated: the Mobile Manipulator aligns the beam horizontally and the Crane lowers the beam into the second stanchion.

3.1.3 The Software

This project employs a multi-layered robot architecture. Each robot, as shown in the following figure, has planning, executive, and behavior layers. Each layer can communicate only with its counterpart on the other robots and the layers beneath and above it on the current robot. Under this scheme, there is no central coordination as each robot communicates with itself and other robots as plans are developed and actions are performed.

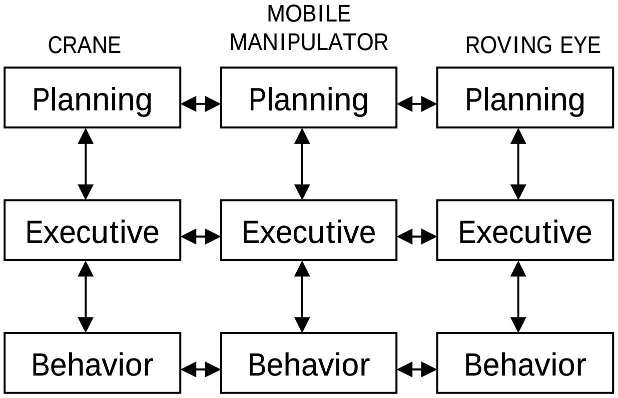


Figure 11 – Each robot, or agent, has three layers: planning, executive, and behavior. The planning layer develops a high level abstract sequence of tasks. The executive layer decomposes tasks into a series of smaller operations based on each agent's skill set. The behavior layer implements the skill set in the form of hardware-level control.

The planning layer is responsible for reasoning about high level goals, sequencing abstract actions, and negotiating with other layers about resources and commitments. In the Trestle architecture, the planning layer would develop a plan similar to (1) dock the first end, (2) move to the second end, and (3) dock the second end. Currently, the planning layer is fixed to the plan suggested. Each agent's planning layer begins with a static role in the plan which is then passed to the executive layer. Consequently, all exceptions are handled in the executive layer and there exists no way to reorder the sequence (e.g., to dock the second end first). Appendix B explores several planners which may hold some promise for a complete implementation of the planning layer (Simmons 2000).

The executive layer receives the abstract plan from the planning layer and decomposes each step in the plan into a sequenced arrangement of behaviors. The execution of these behaviors is then synchronized with the other agent's executive layers. For example, a simplified executive would decompose step (1), docking the first end, into (A) grasping the beam, (B) pulling the beam into the stanchion, and (C) releasing the beam. The executive layer also receives information from the behavior layer about when objectives have been achieved or when progress has failed. Figure 12 shows a detailed view of all three agents' executive layers. Notice that each agent is responsible for maintaining its section of the system task tree.

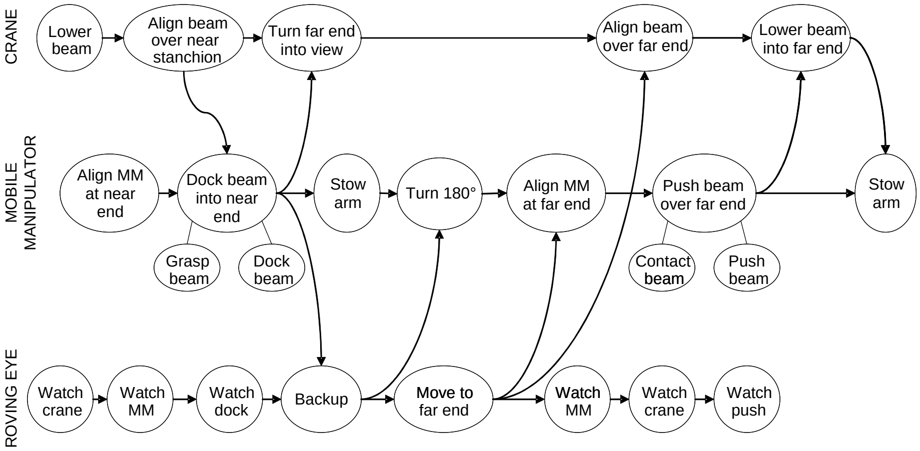


Figure 12 – Task tree that specifies coordination across multiple robots to perform assembly of a beam. Synchronization with the Roving Eye tasks has not been shown for clarity.

The behavior layer is the closest level to the hardware and provides a skill set for each robot. The functions embodied at this layer, known as blocks, are primarily reactive closed-loop control and low-level hardware interfaces. Behaviors may synchronize and exchange data with other behaviors on other robots. For example, the behavior layer would take task (A), grasping the beam, from the executive layer and enable a parameterized version of visual servoing (note tasks (A) and (B) from the executive layer are both visual servoing operations with different parameters). A visual servo operation by the Mobile Manipulator would make a high-bandwidth connection with the Roving Eye across which fiducial pose information would be exchanged. Using the pose information, the Mobile Manipulator would then servo its end-effector through a series of waypoints grasp the beam. Once there, another behavior would be enabled to turn on the electromagnet.

3.2 Sliding Autonomy

The Trestle project seeks to investigate the qualities of sliding autonomy, specifically in terms of increased speed and reliability. The Trestle project augmented the existing DiRA project with sliding autonomy, evaluated the changes, and developed an additional assembly task which relies more heavily on sliding autonomy.

3.2.1 Operator Interactions

It is tempting to desire a system in which every aspect can be controlled under either autonomous or operator direction. A task or behavior is switchable if it can be controlled by either the operator or the autonomous system. When identifying areas that would benefit from sliding autonomy, however, many components within the system did not have meaningful operator modes. For example, the TrackingBlock is part of the Roving Eye's behavior layer and is responsible for identifying fiducials and generating pose information from the stereo images. If lighting conditions were poor or for some other reason the Roving Eye was unable to identify the poses of fiducials, it is conceivable that the operator might wish to assume responsibilities of the TrackingBlock. However, without some significant and very block-specific enhancements, it would not be productive for the operator to attempt to estimate x, y, z, roll, pitch, and yaw values for a pair of fiducials from two stereo images. As a result, switching functionality for blocks with no meaningful operator mode is not provided.

There are, however, many tasks and behaviors whose characteristics make them prime candidates for sliding autonomy. The Mobile Manipulator's VisualServoBlock implements the control loop which, using feedback from the Roving Eye, moves the end-effector toward some goal position. As suggested in the code fragment below, an error is determined between the desired and current positions. If this error is within some tolerance bounds, the executive layer is signaled that the servoing goal is satisfied. If the goal is not yet achieved, commands are issued to other behavior blocks to move the end-effector via RMRC (see Section 3.1.1).

```
error = desired_position - current_position
      if ( error < tolerance )
      then done=true
      else move(error)
```

Fragment 1 – Visual servoing operations use feedback from the Roving Eye to move closer to the goal pose. If the goal is within a certain tolerance distance, the goal is considered satisfied.

Developing a switchable version of the VisualServoBlock enables the operator to intervene on behalf of the Mobile Manipulator during the near end and far end dock tasks. The result is that the operator can implement exception handling during almost any part of the overall task for the Mobile Manipulator.

The Roving Eye's primary goal is to provide position information for visual servoing. Because the Roving Eye is fairly reliable and because many of its tasks do not have meaningful operator modes, relatively few Roving Eye tasks are made switchable. The most notable switchable behavior is the VisualSearchBlock. The VisualSearchBlock is responsible for locating fiducials that have moved out of view. Though the autonomous VisualSearchBlock rarely generates an exception demanding operator intervention, the operator is often able to determine where the Roving Eye needs to look to find a fiducial. To increase the system's speed, the operator can intervene and simply command the Roving Eye to look in the correct direction.

Much like the Mobile Manipulator, the Crane is a prime candidate for a switchable version of its visual servoing behavior. However, because this servoing very rarely fails, a switchable version of the visual servoing is not implemented. See Section 6.2.3 for a related discussion of task-centric approach to control.

3.2.2 Monitors & Actions

An advantage of sliding autonomy is the ability to have the autonomous system monitor the operator and, conversely, to have the operator monitor the autonomous system (Kortenkamp 1997). Generally, when an element of the system is switched from autonomous to operator control, the operator assumes all the responsibilities of that element. Consider the code fragment suggested for the VisualServoBlock (see Fragment

1). If the operator took control of such a block, he would be responsible for both moving the arm and determining when the arm was at its destination (within some tolerance). Under such architecture, there is no mode where the autonomous system determines when the arm is at its destination under operator control.

In an effort to develop a mode whereby operators could monitor autonomous systems and autonomous systems could monitor operators, a distinction between monitor and action system components was identified. A monitor component is responsible for determining when a goal criterion has been satisfied. An action component is responsible for moving the system closer to that goal state. The separation of monitor and action components for the VisualServoBlock is shown in Fragment 2. The monitor component is responsible for determining when the position of the end-effector is within some tolerance distance of the goal. The action component is responsible for moving the arm closer to the goal. This idea of separation is parallel to developing an independent process to constantly monitor the operator's actions (Kortenkamp 1997).

monitor

```
error = desired_position -
        current_position
 If ( error < tolerance )
      then done=true
```

action

```
error = desired_position -
        current_position
move(error)
```

Fragment 2 – Separating the visual servoing operation into monitor and action components allows a much finer variety of sliding autonomy. A monitor block is responsible for determining when the goal is achieved. An action block is responsible for progressing the system closer to the goal.

It is worth noting the code duplication that is apparent after the separation. Two blocks now calculate the same error value between the desired position and current position. The duplication can be alleviated if one block calculates the error and then shares this information with the other block. Sharing is easily done under the system's architecture, but is not shown for clarity (Simmons 1998).

3.2.3 Operator Agent

One result of this thesis is the development of a structured means by which to implement sliding autonomy. In addition to the monitor/action separation discussed previously, the methodology used by the operator to interface with the autonomous agents was carefully chosen. The most natural extension of the architecture suggested in Figure 11 is to wrap the operator processing and graphical user interfaces (GUIs) in an operator agent (see Figure 13). From one perspective, the operator agent acts as the human's representative among the autonomous agents, also known as a liaison agent (Martin 2003).

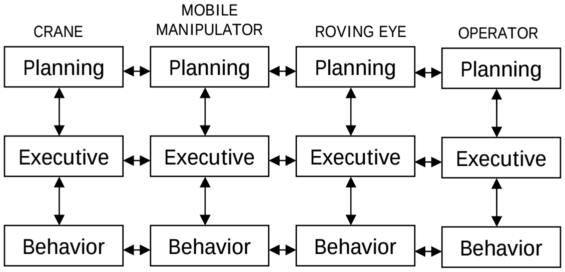


Figure 13 – The operator is represented by an operator agent with analogous construction to a robotic agent. The human operator can interface with any robotic agent at any level.

The operator agent allows the human to affect the planning, executive, and behavior layers of all other agents. Though the current implementation of the planning layer is a static plan, the operator agent would eventually allow the human the means to alter the currently executing plan. At the executive layer, the operator can assist in exception handling or other task coordination. For example, the current system will attempt to dock the end of the beam three times. If the dock fails three or more times the operator and Mobile Manipulator initiate recovery strategies via the executive layer interface. If that recovery strategy involved a teleoperated dock, the operator would provide control via the behavior layer interface.

The separation of monitors and actions offers the system a greater switching flexibility. The operator agent provides a means to represent the operator within the autonomous system. The addition of sliding autonomy to the system involved implementing an operator agent, creating appropriate GUIs within the operator agent, separating monitors and actions, and developing methods to switch between tasks and behaviors.

Chapter 4 The Trestle Experiments

The combination of human and autonomous control has the ability to increase both the speed and reliability of many operations. Preceding chapters have mentioned some of the applications that would benefit from sliding autonomy (see Chapter 1) as well as described an architecture that supports human interaction (see Chapter 3) in the Trestle project. Furthermore, an implementation of a set of switchable tasks that allow the operator to take control at several different stages of the beam assembly task have been described. This chapter describes a set of experiments that compare purely teleoperated, purely autonomous, and sliding autonomous versions of the beam assembly. For each mode of the system, the assembly was run fifty times and data and observations were collected. The following chapter analyzes and draws conclusions from this data.

4.1 Purely Autonomous System

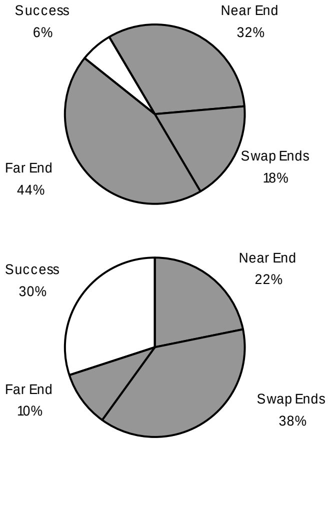
The purely autonomous system is started by a human operator and runs until completion or failure with no further human assistance. As described in Chapter 3, the system's goal is to dock the first end of the beam, turn, and then dock the second end. A run is considered successful when both ends of the beam are securely docked, the robots have disengaged their manipulators from the assembly, and the software indicates completion. A run is considered a failure when the system or a passive, skilled observer determines that it is unable to proceed further. A determination is made only after giving the system ample time to execute any existing exception handlers.

As mentioned, the original intent of this experiment was to run the autonomous process fifty times. After fifty runs, however, an unexpected trend emerged: the rate of success was only 6%. That is, the autonomous system was only able to dock the beam three out of the fifty experiments. As seen in Figure 14 (top), the points of failure were concentrated in the far end docking. Further analysis showed that the cause of the failures was primarily related to the Mobile Manipulator's end-effector controller. Docking the far end of the beam required that the manipulator operate near a singularity where movement became highly non-linear. The autonomous system would be significantly disadvantaged when compared with a sliding autonomous system that simply circumvented this kind of error (see Section 6.2.1 for further discussion). As a result, the arm controller was replaced with the more reliable RMRC controller (see Section 3.1.1).

With the new controller, fifty more trials were attempted. The RMRC controller worked as expected, but new errors emerged and only a 30% success rate was achieved (see Figure 14, middle). Analysis revealed that the swap ends procedure for the Roving Eye was flawed and caused several robot collisions; this problem was remedied. Additionally, an appropriate autonomous handler was created for a commonly occurring exception condition that prevented the far end dock from succeeding. The exception occurred when a slight misalignment caused the beam to become hooked on the docking clamps. The exception handler recognized the condition and prompted the executive layer to repeat the far end docking procedure.

Armed with the new enhancements, the final set of fifty trials resulted in a success rate of 64% (see Figure 14, bottom). Unlike the previous sets of trials, there were no errors that continually plagued the system. Instead, the failures were caused by a variety of events that occurred only a few times. As a result, developing many different autonomous exception handlers would have represented a significant resource investment and would have contributed little to the overall success rate. Listed below are some examples of the errors that occurred.

1. Hardware failures. Several isolated incidents caused the robot hardware to malfunction. In these cases it might not be possible to continue the beam assembly, but the system should at least recognize that a malfunction has occurred.
2. Software failures. Software crashes and bugs contributed to several unsuccessful runs. Though perhaps subtle, there are clearly autonomous fixes for these kinds of errors.
3. Visual perception errors. The Roving Eye provides the positions of fiducials using a stereo pair of cameras. This system is subject to error and, in rare cases, can cause robots to move erratically. As a result, fiducials can be obscured (e.g., the Mobile Manipulator blocks the Roving Eye's view) and robots may collide.
4. Others exceptions. Often, the beam will slip from the Mobile Manipulator's grasp. An autonomous exception handler was developed that realizes the condition and attempts to re-grasp the beam. During one trial, repeated execution of the exception handler still failed to grasp the beam – that is, the exception handler failed. In another instance, part of the assembly broke apart. The system had completed the near end dock, turned to the second end, and the first end then became undocked. The system had no way to identify the contingency or to develop a recovery strategy. An autonomous handler would require greatly increased sensing and replanning capabilities.



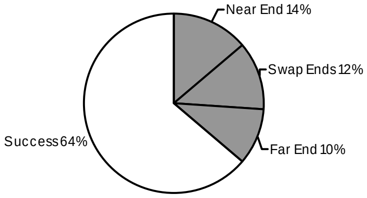


Figure 14 – Several data sets were collected for the completely autonomous system. The initial (top) and secondary (middle) sets were each plagued by a few particular errors. These errors represented flaws in the autonomous design, and sliding autonomy would have unnaturally benefited both of these systems. The errors were fixed and a final set of autonomous data was taken (bottom) in which the errors that did occur were spread evenly across the task and of a wide variety. Shaded wedges were failure runs, white wedges are successful runs.

In addition to the 64% success rate, the autonomous system had an average completion time of about 9.9 minutes with a standard deviation of 1.6 minutes. These numbers will have more meaning when compared to the performance under different levels of sliding autonomy in the next sections.

4.2 Purely Teleoperated System

In teleoperated mode, four users were asked to complete the beam assembly task a total of fifty times. All users were experienced with robotics, but were not necessarily experts with the particular task. Users were presented with a collection of robot-specific interfaces that allowed them to individually and directly control each robot. These interfaces allowed the operators to move end-effectors (in position and velocity modes), reposition robots, and manipulate and sense the environment as each robot allowed. The primary feedback was provided by the left and right stereo images from the Roving Eye. These two images were displayed on the screen alongside the user interfaces.

It is worth noting that the experimental results from the teleoperated system will be dependent both on the skill of the operators and the quality of the user interface. Highly skilled operators may be able to operate the system more efficiently and a well developed interface may facilitate every user's performance. As a result, these results represent a single point of possible teleoperation results.

The human operators were able to successfully dock the beam 96% of the time, much more reliably than the autonomous system (see Figure 15). On the other hand, the average time for teleoperation was 12.5 minutes with a standard deviation of 4.0 minutes. The users were much less timely with their completions. From a more qualitative standpoint, the operators were adept at gross robot maneuvers such as the swap ends. Operators had much greater difficulty performing fine manipulation tasks such as docking the beam, where tolerance was low.

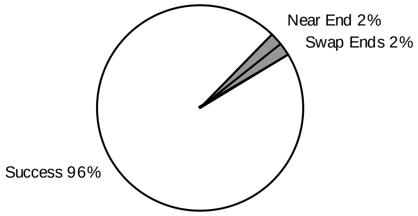


Figure 15 – Teleoperation Results. Human operators were able to successfully dock the beam 96% of the time, significantly higher than the autonomous system.

Reponses from operators suggest that the left and right stereo images did not provide sufficient disparity and made depth perception difficult. Additionally, the narrow view of the Roving Eye created ‘tunnel vision’, and forced the users to rely on their memories to maintain overall task cognizance. Chapter 7 describes plans to account for these deficiencies in upcoming experiments.

4.3 Sliding Autonomous System

Finally, the system which allowed the full range of sliding autonomy was tested. The implementation details as well as a description of switchable tasks are provided in Section 3.2. The system was always started autonomously and there were no cases of scripted or robot-initiated mode changes. All cases of intervention were human-initiated, as the operator stepped in and tried to optimize the system performance whenever possible. When a mode switch was initiated, the operator was presented with a user interface specific to the current task. For example, if the operator intervened during a visual search operation (when the Roving Eye attempts to reacquire lost fiducials, see Section 3.2.1), the operator was presented with a list of the missing fiducials. The operator could then take control of the Roving Eye and cameras as necessary.

For feedback the operator could again use the left and right camera images from the Roving Eye. Additionally, the operator could make use of the output of any of the autonomous processes. The most useful output was the relative positions of fiducials. Although the operator had difficulty making significant use of millimeter precision, the information did help compensate for the lack of depth perception from the camera images.

As show in Figure 16, the overall autonomous system completed the beam assembly task successfully during 94% of the runs. The average completion time was 9.9 minutes with a standard deviation of 1.9 minutes. The completion rate compares well with the teleoperated system and the completion times compare well with the autonomous system.

To understand these results further, successful runs are classified as having either discretionary intervention or mandatory intervention. A run with discretionary intervention is a trial in which the operator contributes mildly or not at all to the system. As seen in Table 1, the number of discretionary successes is comparable with the autonomous system (this is anticipated, as the operator added mildly and did not affect the success/failure outcome of the run). Although the operator’s contribution did not affect the outcome, it did serve to speed up the autonomous system. For example, the operator may have helped to find a missing fiducial or signal the completion of a waypoint. Other successes were classified as having mandatory intervention. In these runs, the operator contributed significantly and changed a potential failure into a success. The increase in successes from autonomous to sliding autonomous systems is a result of these mandatory interventions. There was, however, a time penalty as the operator now had to recover from a variety of errors previously left as failures.

	Successes	Completion Times	
		Mean	Std-dev
Fully Autonomous	64%	10m	1.5m
Sliding Autonomy	94%	10m	2m
Discretionary Only	68%	9.5m	1.5m
Mandatory Only	26%	11m	2.5m
Teleoperated	96%	12.5m	4m

Table 1 – Overall Results. The sliding autonomous system combined the speed of the autonomous system with the reliability of the teleoperated system.

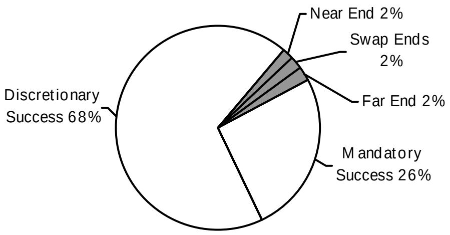


Figure 16 – Sliding Autonomy Results. The sliding autonomous system's overall success rate was 94%, nearly equivalent to the teleoperated system.

The autonomous system was best at completing the assembly task in a timely fashion. The teleoperated system was best at completing the assembly task reliably. By adjusting the level of autonomy, the operator was able to maximize both the speed and reliability of the system when sliding autonomy was enabled.

Chapter 5 System Modeling

Sliding autonomy has the potential to improve both the speed and the reliability of a system. This thesis explores these ideas in the context of an assembly task and develops more general conclusions from the results. It is also possible to apply more formal methods of system analysis and draw similar conclusions. In this chapter, Petri Nets (PN's) are used as a means to model the way in which sliding autonomy increases system reliability and speed.

A system that includes multiple coordinated robots with human interactions is difficult to analyze. The autonomous system will complete the task according to some performance distribution depending on initial conditions and external exception events. Likewise, human controllers will operate and contribute according to some analogous distribution depending on communication delays, operator burden, and skill level.

A PN can be used to visualize complex systems. As related in Appendix A, PN's can be used to model actions performed in parallel or sequentially and develop notions of resource allocation. PN's can model a variety of other behaviors, can be applied in discrete and continuous domains, and can describe deterministic and non-deterministic effects. In addition to visualization, some PN's can be converted into finite state machines and employ finite automata analysis techniques.

5.1 Increasing Reliability

The purely autonomous beam assembly task might be represented with a simple PN as shown in Figure 17A. When P0 is marked, the system is waiting; the firing of T0 corresponds to the beginning of the task. When P1, P2, or P3 is marked, the system is performing the first end, swap ends, or second end dock, respectively. T1, T2, and T3 correspond to the successful completion of their associated task (e.g., T1 corresponds to the successful completion of the first end dock).

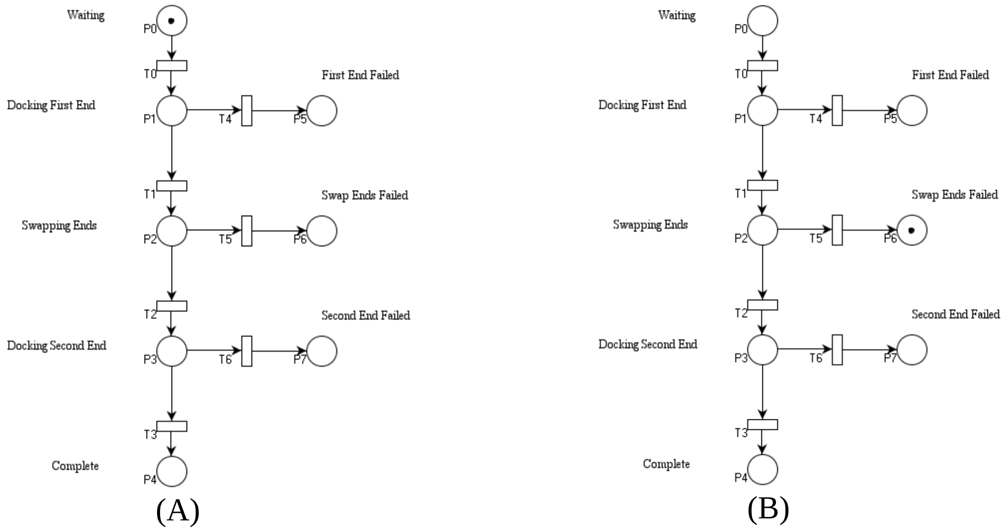


Figure 17 – (A) The autonomous beam assembly task in its initial state. If no failures occur, the transition path is {T0, T1, T2, T3}. (B) However, the assembly task may fail. In the case shown here, the task has failed during the swap ends and arrives in a deadlocked state.

Ideally, the system will follow the transition path {T0, T1, T2, T3} and perform a successful completion. However, as has been noted, the system is likely to fail during each step. Transitions T4, T5, and T6 relate to failures at the first end, swap ends, and second end dock procedures, respectively. If the swap ends were to fail (i.e., transition T5 fires), the system would find itself in a condition where no action could be taken to reach the goal (marked successful completion in P4). This condition is known as a deadlocked state. There is no action that the system can take to move itself out of a deadlocked state. In much the same way, the autonomous system may encounter exception conditions beyond its recovery abilities. The PN model could easily be scaled to include more tasks with many different types of failures. There are algorithms to detect deadlock states (Chen 1995).

To improve reliability, the goal of sliding autonomy is to eliminate deadlocked states. That is, to improve reliability, the system's overall exception handling capability must be enhanced. The conceptualization is straightforward: tasks that lead to anticipated deadlock states must be switchable (i.e., have an operator mode). Each previously deadlocked state now has an enabled transition back to the main task in the sliding autonomous PN in Figure 18. Transitions T7, T8, and T9 correspond to the operator taking control of the associated task and recovering from the exception. For example, if the system fails during the swap ends because the Mobile Manipulator fails to turn, the operator may intervene (transition T8 fires) and return the system to a stable mode (marked P2).

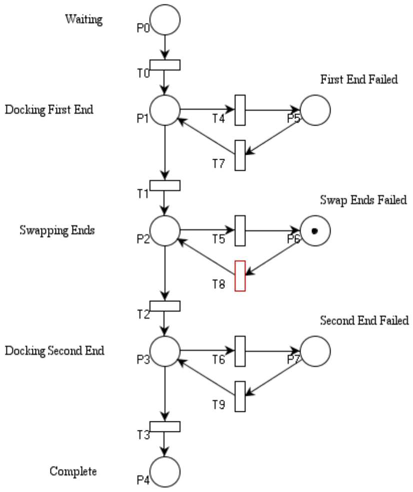


Figure 18 – With sliding autonomy, the autonomous system proceeds normally when there are no failures via transition path {T0, T1, T2, T3}. When an failure occurs (a swap ends exception is represented here), the operator can intervene and return the system to error-free operation. The operator is employed only when needed and there are no exception deadlock states.

5.2 Increasing Speed

The general idea behind achieving a speed increase is that the system is given an alternative means to complete a task in a more efficient way. An increase in speed was readily apparent during the VisualSearchBlock. As described in Section 3.2.1, the VisualSearchBlock is responsible for finding fiducials that are no longer in the Roving Eye's field of view (FOV). Assume that the VisualSearchBlock can always find fiducials and that there is no deadlock state. However, depending on how well the missing fiducial is hidden, the search procedure may take different amounts of time. Using a PN analysis, this section demonstrates that operator assistance can accelerate the autonomous search process. Furthermore, the amount of acceleration is dependent on the ability and availability of an operator. When the operator is adept and available, the speed improvement can be substantial. When the operator is inept or inaccessible, the speed of the system degrades to the autonomous performance level.

The PN in Figure 19A represents a search procedure. Two fiducials (markings in P1) are currently being tracked by the system (that is, the system knows where they are and they are *found*). At any time, a fiducial may become *obscured* by an object (transition T1), possibly by the Mobile Manipulator's end-effector. Because the system cannot move the arm until it finds the fiducials, the Roving Eye must attempt to reposition itself for a better perspective (transition T3). On the other hand, sometimes a fiducial moves just beyond the Roving Eye's FOV (transition T2) and is *missing*. In such cases, a quick movement of the Roving Eye's cameras, using the pan-tilt unit (PTU), can reacquire the missing fiducial (transition T4). The movement of the entire Roving Eye (transition T3) takes significantly longer than the movement of the cameras (transition T4). Thus, there is a much higher time penalty when the system loses a fiducial because it is obscured.

Notice that the arcs from P3-T1 and P2-T2 are inhibitor arcs because both short and long term searches cannot proceed at the same time.

Interpretation of Visual Search PN (Figure 19)

P1 – Markings indicate the number of fiducials that are currently being tracked (i.e., fiducials that are found).

P2 – Markings indicate the number of fiducials that are currently obscured.

P3 – Markings indicate the number of fiducials that are missing (i.e., just beyond the FOV).

T1 – Firing corresponds to the obscuring of a fiducial.

T2 – Firing corresponds to the losing of a fiducial.

T3 – Firing corresponds to the identification of an obscured fiducial. This often requires slow Roving Eye movements.

T4 – Firing corresponds to the identification of a missing fiducial. This often requires quick Roving Eye movements.

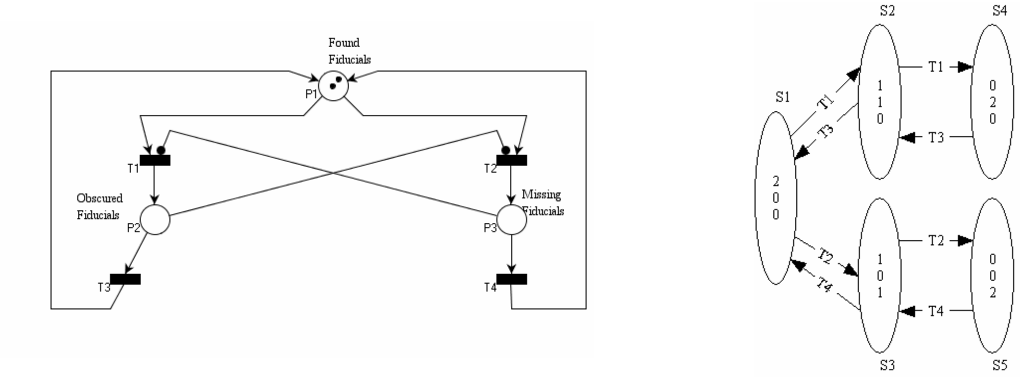


Figure 19 – (A) A PN for the autonomous visual search procedure. When fiducials are obscured, the recovery process takes considerable time (T3). When fiducials are missing, the recovery process is usually timely (T4). The state space representation (B) shows the different markings of the system. Only one marking, S1, describes the case when both fiducials are found.

With a stochastic PN, some timing characteristics can be assigned to the system. If the timings are modeled with an exponential distribution (see Appendix A), a mean time between firings can be assigned to each the transition. A transition will fire when it is enabled and with some probability related to the time it has been enabled since the last transition fired. With this information, it is possible to develop a probability distribution of the likely states. Recall that the state of a PN is a vector describing the markings of every place.

The state diagram for the system is shown in Figure 19B. Each transition between states is associated with a firing of a transition. Because these firings are randomized functions which follow an exponential law, the system can be analyzed as a discrete Markov process. It is then possible, using the techniques outlined in Appendix A, to determine the probability that the system is in a certain state. For this system, the interesting state is S1, when both of the fiducials are found (two marks in P1). A high probability for S1, $P(S1)$, means that the system is likely to know the locations of both fiducials. It is possible to solve analytically for $P(S1)$ in terms of the mean delays of the transitions, $\{t_1, t_2, t_3, t_4\}$. However, the result is rather tedious and offers little insight. Instead, values are assigned to the transition times as relative estimates of the actual process. Only the relative magnitudes, not the values themselves, are important. For example, as has been discussed, the mean firing time of T3 should be larger than T4 because it takes more time to find the obscured fiducials. Let $\{t_1, t_2, t_3, t_4\} = \{2,2,4,0.1\}$. That is, on average a fiducial is obscured or goes missing every 2 minutes, it takes 4 minutes to find an obscured fiducial, and it takes 1/10 of a minute to find a missing fiducial. Given these numbers, both fiducials will be found $P(S1)$=10.9% of the time by the autonomous system. During the remainder of the time, the system will not know the location of one or more fiducials.

The system is plagued by the poor performance of the search when a fiducial is obscured (the recovery procedure via T3 is costly). Operator intervention can help because the human can often immediately decide where and how to position the Roving Eye. That intervention would constitute sliding autonomy because now the system can perform the task both autonomously and through teleoperation. A PN representing this behavior would include an alternate method of finding the obscured fiducial via T5 (see Figure 20). The operator is used as a resource, represented in P4. If the operator can locate the obscured fiducial in an average of $t_5 = 0.1$ minutes, the probability that both fiducials are found at a particular instant is 82.8%. Because the system spends less time looking for obscured fiducials, it is more likely to be in the state where both fiducials are visible.

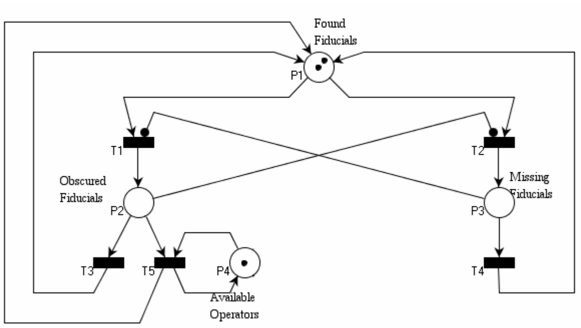


Figure 20 – The addition of an operator in the now sliding autonomous system allows the task to be accomplished more quickly.

The addition of an omnipresent operator in Figure 20 is a trivial case. Clearly if there is a faster way to do a slow task, the system will perform faster. A more interesting analysis arises when the operator's availability is also random (see Figure 21A). The operator may be busy with other robot teams or subject to communication latency. This case is similar to what might be encountered where an operator is teleoperating multiple sliding autonomous teams. When the operator mark is at location P4, the operator is available to help find obscured fiducials. When the operator mark is in P5, the operator is in a communications blackout. Transition T6 is fired when the operator goes into a blackout period and transition T7 is fired when the operator returns from blackout and is again available. The state diagram is show in Figure 21B; notice that there are now two states where both fiducials are found (depending on whether or not the operator is available). When both fiducials are found, it is unimportant whether or not the operator is available. $P(F)$ is the probability that both fiducials are found at a particular instant: $P(F) = P(S1) + P(S4) = 1 - P(\overline{F})$.

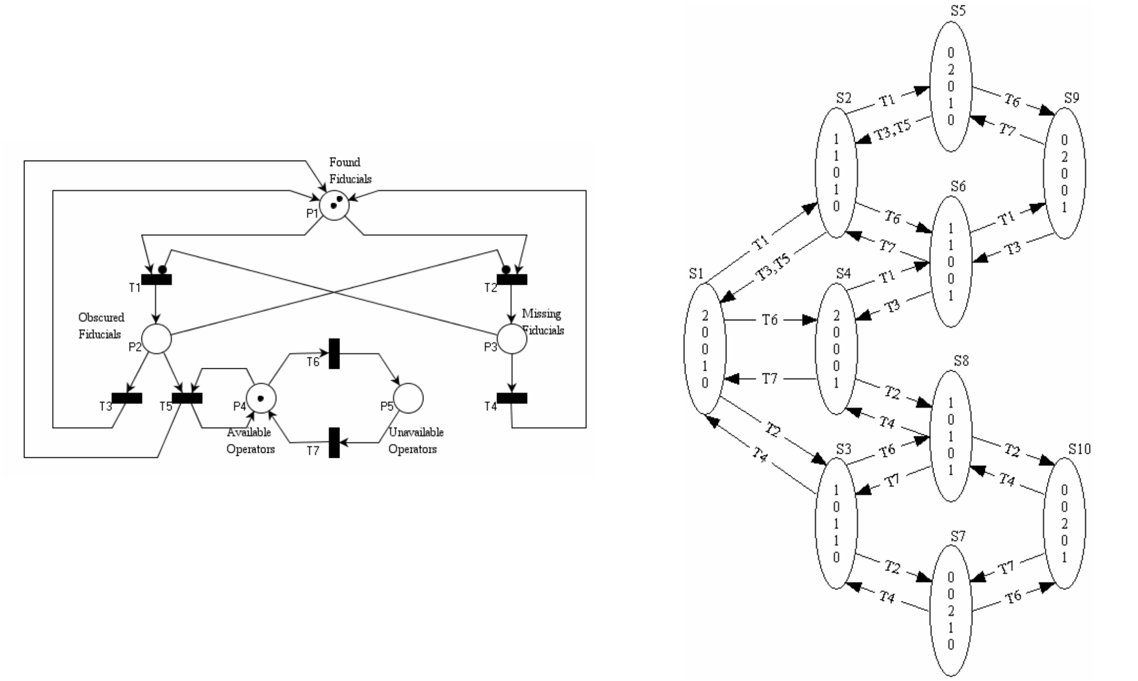


Figure 21 – (A) The operator is now subject to communications blackouts and may be unavailable (P5) to assist the system (T5). (B) Simple changes in the PN can represent complex changes in state space.

At some point, the communications delay will be sufficiently long and often enough that the operator will no longer be able to increase the system's speed (and increase the probability that the system can see both fiducials). Assuming the same timing characteristics used previously, a system is constructed where the mean time to blackout is $t_6 = {}^{1}\!/_{\tau}$ and the mean time of a blackout is $t_7 = \tau$ (the single variable τ is used to simplify the analysis). As τ is increased, the blackouts will become more frequent and longer. As shown in Figure 22, $P(F) \geq 50\%$ when $\tau \leq 1.47$ minutes. In other words, when the mean time between blackouts is greater than 0.68 minutes and the mean time of a blackout is less than 1.47 minutes, the operator can help the system to have both fiducials visible the majority of the time. The shape of the curves in Figure 22 is dependent on the choices made for t_6 and t_7.

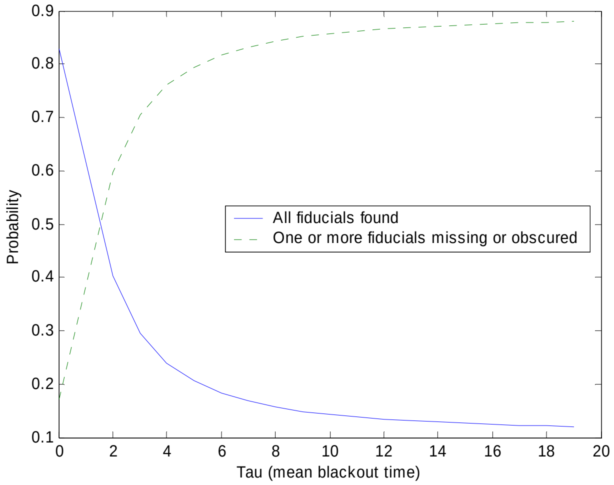


Figure 22 – As the frequency and duration of operator unavailability increase (e.g., due to communication blackouts), the system is less probably in a state where both fiducials are found.

It is interesting to note that in Figure 22, as the operator becomes increasingly less available, the system operates with the same efficiency as the purely autonomous system (about 10.9%). The use of PN can help visualize the means by which sliding autonomy can increase a system's speed. Additionally, by assigning timing characteristics to transitions, it is possible to analyze the system. In this case, the system was analyzed to determine what minimum mean operator attention was necessary to keep the system in the goal state the majority of the time.

Chapter 6 Trestle Results

Building on the preliminary analysis in the previous chapters, this chapter attempts to draw more qualitative conclusions. Satisfaction of the thesis' goals of increased speed and reliability are argued, and several additional results that emerged as a result of the experiments are discussed.

6.1 Thesis Goals

The goal of this thesis is to develop an architecture for sliding autonomy that allows a system to harness the best speed and reliability qualities of operator and autonomous control. By harnessing the speed of the autonomous system and only employing the operator when necessary, the system's overall speed is improved. Simultaneously, operator control allows the system to handle a much wider variety of exceptional conditions, simplifies the way the system handles those exceptions, and results in greater success rates.

The sliding autonomy architecture proposed depends both on the details of implementation (i.e., the manner in which the software supports operator interaction) and the process by which adjustable tasks are selected and switched (i.e., the types of tasks with which the operator can interact and the timing of those interactions). When autonomous and teleoperated versions of the system are compared, the human operators are able to complete the task more often, but they generally require more time. It is not necessarily the case that every system will demonstrate this modality, as suggested in Section 5.2. If the autonomous system is continuously both faster and more reliable than teleoperated control, there is no justification for sliding autonomy. Similarly, if teleoperated control always demonstrates better characteristics than autonomous control, there is no reason to support sliding autonomy. In general, however, we conclude that autonomous and teleoperated control will each be most appropriate for some set of subtasks and, furthermore, that these sets will be dynamic. In an expanded version of the beam assembly, for example, it is most efficient (in fact, necessary) for the operator to move the robots into their initial positions. This initialization task is best suited to teleoperated control. On the other hand, the actual docking of the beam into the receptacle is done faster and more successfully by the autonomous system because the autonomous system can make more accurate use of the visual feedback. The dock, however, may be better done by the operator when exception conditions occur. If the beam becomes wedged in the receptacle during a dock, the autonomous system may become unsuitable for the task and operator intervention may be required. In this and similar cases, it is necessary for a system to support the ability to switch tasks between autonomous and teleoperated control.

For the reliability and speed increase results to be general, several conditions are necessary. First, tasks that can generate unpredictable exception conditions (and supporting tasks) must be switchable. In this way, the most appropriate exception handler (often the operator) can provide assistance where needed. Second, any task that can be done more quickly by the autonomous system or the teleoperator must be switchable. This switching will allow the most timely agent (teleoperator or autonomous) to complete the task. Third, either the autonomous system or the teleoperator must have the ability to switch tasks when either exceptions or the potential for speed increase exist. This functionality will give the overall system the means to improve its speed and reliability. It is worth noting that in this system there was no communication delay and the operator's sole task was to monitor system performance. In such instances, the operator can be primarily responsible for switching a task's mode. If significant communication delays exist or if the operator's attention is divided, the responsibility to shift modes may be placed more on the autonomous system.

The separation of monitors and actions (as discussed in Section 3.2.2) proved useful throughout the sliding autonomous trials. Often the autonomous block responsible for determining when an object had arrived at a goal location (i.e., the VisualServoBlock monitor component) was too restrictive. Visual perception errors would prevent the autonomous system from signaling the satisfaction of waypoint goals. In these cases, the operator could assume control of the visual servo monitoring and force the system to complete visual servoing.

The addition of the operator agent proved to be a useful part of the architecture, as it provided a formal means for the operator to interact with other agents at the planning, executive, and behavior layers. Although convenient, it is not proposed that the operator agent is essential to a productive implementation of sliding autonomy. Instead, the operator agent served as an organizational agent to separate the user interfaces from the autonomous agents. This organization was particularly valuable avoiding the temptation to improperly, but more conveniently, incorporate operator interfaces within autonomous modules.

6.2 Additional Discoveries

While implementing the architecture for sliding autonomy and performing experiments, several trends emerged that suggested new facets to sliding autonomy. Although these topics do not constitute a central component to the thesis statement, the topics have value and suggest alternative uses and considerations for sliding autonomy.

6.2.1 Rejecting Sliding Autonomy as a Panacea

As suggested in Section 4.1, several iterations on the autonomous system were performed before comparing success rates and choosing what tasks to make switchable. Specifically, the initial experiments with the autonomous system suggested success rates of only 6%. Several errors continually plagued the system and made success a low probability event. Because the system was hardly a triumph, sliding autonomy could only help improve the performance and comparing sliding autonomy at this point would not produce valuable results. Conversely, if a system existed with a 100% success rate, there would be no need for any kind of operator intervention. Indeed, the very purpose of sliding autonomy is to help a deficient autonomous system handle a wider variety of situations (and thereby improve the success rate). It is unclear where the addition of sliding autonomy is valuable as an exception handling tool and where it is simply a quick fix, or “band-aid,” for a poorly designed autonomous system.

The solution, we conclude, can be found by studying the types of errors that the autonomous system experiences. In general, as more resources are invested in an autonomous system, its performance (speed and reliability) will improve (see Figure 23). At some point, however, the rate of reward per resource unit will begin to decrease.

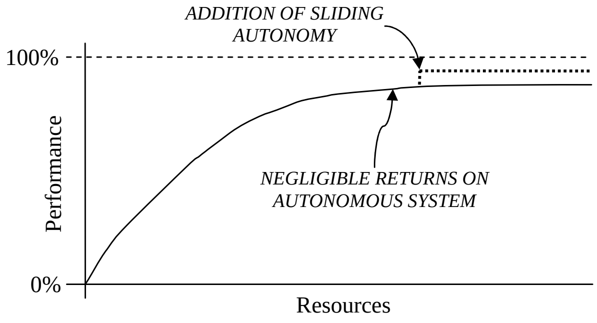


Figure 23 – A typical autonomous system will reach a point where continuing to invest in the system (i.e., increasing resource allocation) will result in a diminishing rate of return in performance. Due to unpredictable circumstances, the autonomous system will never be perfect. Although sliding autonomy may not be able to compensate for every potential circumstance, it can provide a sizable performance increase with little associated resource cost.

A cost-benefit analysis provides one model. It is worthwhile to continue to enhance autonomous capabilities until the cost of enhancements (i.e., the resource cost) begins to exceed performance benefits. The associated costs are, of course, application specific and may depend on risk-adversity and the practicality of operator assistance. A substantial communication delay, for example, may require a system capable of high interim-term performance when the operator is unavailable.

Another perspective suggests that sliding autonomy can circumvent the asymptotic nature of the performance curve in Figure 23. At some point, further investment in the autonomous system yields no improvement in performance. These diminishing returns result because every contingency cannot be planned and handled with certainty. That is, in every situation there will always exist some conditions and failure modes for which the system cannot be fully prepared. The solution, then, is to advance the autonomous system to the point were negligible performance returns are yielded. The addition of sliding autonomy, then, results in a discontinuous jump, improving performance with minimal cost. Although sliding autonomy can boost the system’s ability to handle unexpected conditions, it cannot be assumed that it will make systems indestructible.

6.2.2 Autonomy in Manual Control

In Section 4.2, it was noted that the performance of a teleoperated system will be dependent on the skill level of the operators and the quality of the user interface. If the operators are well practiced and very familiar with the task, they will probably perform better than a novice user. One common complaint from the operators was that the left and right camera images from the Roving Eye did not provide sufficient disparity to judge the scene’s depth (see Section 4.2). When depth perception was lacking, the more experienced operators used other cues to guide their movements (e.g., shadows or a precisely chosen camera angle). Less experienced users took longer to develop such intuition.

In the end, the distribution of successes and times for the four differently skilled users was relatively similar (see Figure 24). However, the issue arises as to what level of user interface is best, as control is still considered to be purely teleoperation. One possible method to alleviate the lack of depth perception is to provide the user with a 3D reconstruction of all robots and objects in their current positions. Such a reconstruction would require that the module responsible for calculating the poses of fiducials – an autonomous module – be enabled during teleoperated control. Enabling such a module would seem to violate the goal of pure teleoperation. On the other hand, a button labeled “dock the beam,” which the user must press and results in the completion of the task, does not seem to qualify as teleoperation.

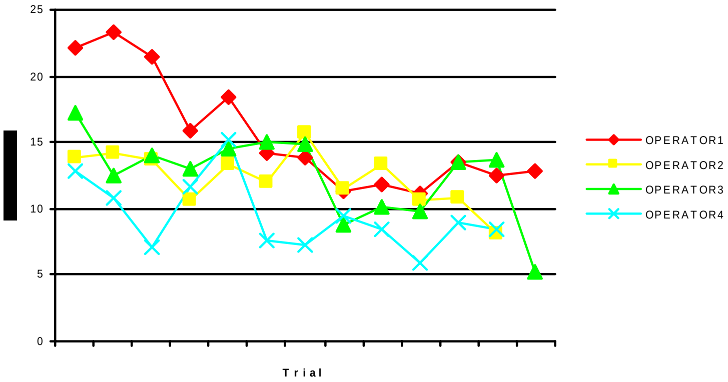


Figure 24 – After some experience, the completion times for the four users during teleoperation show that the users are comparable in skill level. Most users exhibited a slight learning curve as their intuition about the system developed.

In the end, we made the subjective distinction that, in order to maintain a teleoperated system, the operator must remain in the higher levels of the control loop. For the arguably trivial case of the “dock the beam” button, the outer most control loop is completely autonomous. For the 3D reconstruction, the operator is analyzing the 3D data and determining how to move objects. The operator, thus, is very much a part of the control loop. Using the output of autonomous modules to determine the action of autonomous input modules, when done by the human operator, is still teleoperation.

6.2.3 Task-Centric Control

Many implementations of sliding autonomy allow the operator to interface through a variety of means. Typically, however, they reduce to an operator controlling a single robot or an aspect of a single robot (e.g., Doggett 1996, Ward 2003, Kortenkamp 1999, Miller, Gillespie 1999, Wannasuphoprasit 1998, and Fong 2001). Part of the Trestle project is to develop strategies for controlling coordinated teams of robots performing tasks that no single robot could accomplish on its own. The project statement guarantees that the robot team will have to achieve goals that require one or more robots to work simultaneously. By extension, then, the operator must be able to teleoperate multiple robots concurrently.

In general, direct control of several robots simultaneously while attempting to achieve some related goal will place a significant burden on the operator. For example, during the dock of the second end of the beam, the Crane and Mobile Manipulator must work together to position the beam (see Figure 25). For the first end dock, the Mobile Manipulator and Crane can work sequentially as the Crane brings the beam into position, stops, and then allows the Mobile Manipulator to precisely dock the beam. The second end, however, requires that the Mobile Manipulator and Crane servo together to dock the beam.

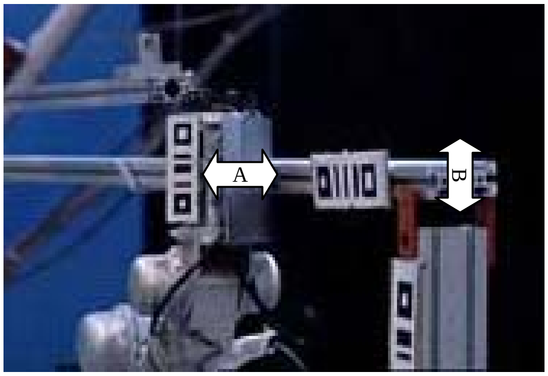


Figure 25 – The Crane and Mobile Manipulator visually servo simultaneously to dock the beam in the second receptacle. The Crane is responsible for controlling movement in the vertical direction (B) and the Mobile Manipulator pushes on the beam in the horizontal direction (A). Separating the axes of motion allows each robot's servo loop to operate independently, but requires careful concurrent control.

The second end dock is a task that is subject to failure and benefits from the addition of sliding autonomy. However, requiring the operator to servo the Mobile Manipulator and Crane together is unacceptable. The ideal solution is to allow the operator two modes of control: (1) servo the beam or (2) servo the Crane and Mobile Manipulator individually. Regarding the former, during teleoperated control of this task, the operator specifies desired movements of the tip of the beam relative to the receptacle. The autonomous system calculates required Mobile Manipulator and Crane movements from desired beam movements. From the operator's standpoint, the task is abstracted to the pertinent goal level – docking the beam – and not the intermediate goal of moving the robots. The latter option is still necessary should exceptions occur within the autonomous calculations.

The Trestle project implements a slightly simpler version of the ideal solution. When the Mobile Manipulator and Crane must work together at the second end, the Crane is slaved to the Mobile Manipulator. When the operator moves the Mobile Manipulator, the Crane follows accordingly. A better implementation would slave the Mobile Manipulator and Crane to the operator's beam movements (as described earlier). We suggest that, in general, the goal during teleoperation is to concentrate the operator's interaction toward satisfying the goals (e.g., beam placement) rather than achieving indirect sub-goals (e.g., robot positions).

6.2.4 Maintaining State

If a naive system were to switch between teleoperated and autonomous control, the autonomous system might lose context (state) and not be able to resume when teleoperation is complete. For example, assume that the operator has intervened during the beam assembly's turn step (movement from first to second dock locations). Once finished, the operator wishes to cede control back to the autonomous system for the second dock. If, however, the autonomous system has failed to realize the operator's actions (the turn) and failed to "remember" its earlier actions (the first dock), the system will not be in the proper state to perform the second dock.

There are several aspects to a mode change from operator to autonomous control. First, the autonomous system must obtain knowledge about what actions the operator performed. This knowledge could result from operator input; the operator could signal task completion through monitors (see Section 3.2.2). In some cases, the autonomous system may be able to recognize that the task is complete or that the preconditions for the subsequent task are met. A precondition for the second dock may be that all relevant fiducials are visible by the Roving Eye. If the Roving Eye detects all these fiducials, the system properly assumes that the second dock can proceed. For more complicated actions, the system may need to better query the state of the world.

Second, after determining what the operator has done, the autonomous system may need to remember what tasks it has already performed (e.g., if the current world state is dependent on previous world states, which may be the case if the entire world state cannot be directly observed). The Trestle system maintains this state indirectly through the task tree. In Figure 26, if the operator assumes control of task (B), the autonomous system still "remembers" what it has done (task (A)) and, thus, what needs to be done next (task (C)).

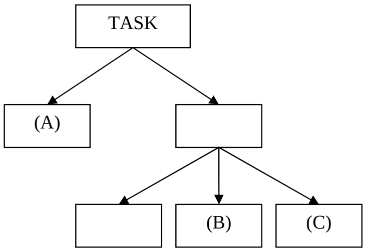


Figure 26 – The Trestle software decomposes tasks and stores the representation in a task tree. When a task is switched between operator and autonomous control, the task tree maintains the autonomous system's state.

In this way, the task tree maintains the state of autonomous system. It may be necessary to regenerate the task tree if the state of the world deviates from the task tree's sequential transitions.

6.2.5 Sliding Autonomy for Debugging

In general, software development is greatly eased with use of a debugger. Debuggers allow a programmer to step through lines of code, examine variables, inspect outputs, and monitor program flow. The result is that a programmer can isolate potential problem areas from the rest of the system. While developing and testing the Trestle system, it became clear that sliding autonomy offered beneficial debug-like functionality.

For example, monitors in the VisualServoBlock (see Section 3.2.2) provided programmers with a formal means to step through waypoints during visual servoing. The programmers could assume control of the monitors or actions and evaluate – in real time – what the autonomous system was attempting to do. Though the same information is available after the run in logs, it is often difficult to remember the physical system's state. Similarly, if the autonomous system is servoing an end-effector, the position of the end-effector is constantly changing, making it difficult to perform any kind of steady state analysis. Pausing the system gives the operator the ability to perform these analyses.

Software debugging is ideally used to perfect code before deployment. In fact, sliding autonomy goes beyond those abilities and can enable imperfect system deployment. In cases where a system must be deployed quickly or before the technology exists for a completely autonomous system, the addition of sliding autonomy can efficiently "fill the gaps." Chapter 1 describes a potential use for sliding autonomy during repair of the Hubble. Robotic technology is not sufficiently advanced to support a system of completely autonomous robots. It is unrealistic to hope for robots that could locate, interface, repair, and test the aging telescope without operator assistance. With sliding autonomy, however, a system could be deployed where teleoperation brings the robots into some difficult position, the robots autonomously replace some delicate component, and teleoperation facilitates testing and dismounting from Hubble. Difficulties inherent in the costs, both human and physical, in operation and maintenance of existing service strategies make this an appealing possible alternative. Should technology later become available to automate more of the process, teleoperated modules can be replaced with autonomous modules.

When implemented properly, sliding autonomy has the potential to combine the best qualities of speed and reliability that the teleoperated and autonomous systems can offer. The implementation requires that modules, where either the autonomous or teleoperated system may benefit the system, be switchable. Additionally, both the autonomous agents and operators must recognize and capture opportunities to perform the mode switch.

Chapter 7 Future Trestle Work

The set of Trestle experiments described in previous chapters provide several useful insights to the value of a system capable of sliding autonomy. There are, however, several weaknesses in the experiments involving the way the human operator was monitored. We believe the human operator, as well as his user interface, needs to be more carefully evaluated during the trial runs. Additionally, there are several aspects of sliding autonomy which cannot be fully explored with the current beam assembly task. This chapter will describe the means by which attempts to better evaluate the human operator's status and future experiments.

7.1 Experimental Limitations

The teleoperation and sliding autonomy results described in Chapter 4 all depended on the performance of a human operator. The addition of this human assistance noticeably helped increase the reliability beyond the autonomous system. The Discretionary Intervention successes resulted in a small average speed increase, but the Mandatory Intervention successes took much longer to complete. This was because a greater burden was placed on the operator and it is this notion of burden that the current results do not reflect.

Data collected during the experiments provides measures for how often and how quickly the different systems could complete a given task. However, in both the teleoperated and sliding autonomous cases, where human assistance is involved, there is no direct measure for the operator's workload. The idea of workload might be useful if the operator were controlling many teams of robots simultaneously. A low-workload system would allow the operator to multitask efficiently. A high-workload system would require the operator's exclusive attention. Similarly, a system with large communication delays (e.g., remote planet exploration) would only be able to place a limited workload on the operator. Such a system is analyzed with PN's in Section 5.2.

One possibility is to measure the amount of data sent to and from the operator. This kind of operator I/O would provide a quantitative metric for how many commands the operator sends relative to how much information he requires. A data rate analysis would be particularly useful when bandwidth limitations existed. It is not immediately evident, though, that the number of bytes exchanged between the autonomous and teleoperated system correlates to the amount of effort the operator expends.

Another more promising possibility is to utilize NASA's Task Load Index (TLX) survey. The TLX survey is designed to be administered to human operators after performing some action. By asking them to rank their experience, the survey attempts to score the demands (as shown in Figure 27) that operators experienced while working. The tallied results can be used to evaluate other aspects of the operator's experience beyond time and success measurements.

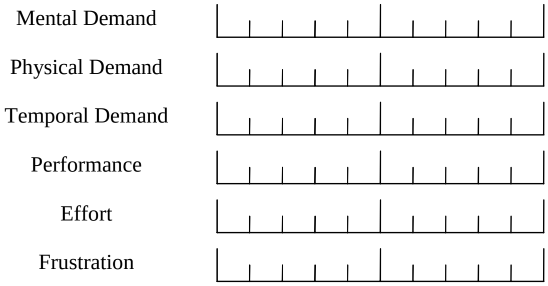


Figure 27 – The TLX survey provides a means to evaluate operator workload during some operation. Such workload information can valuably augment timing and reliability measurements.

Future comparisons of teleoperation and sliding autonomy also require a more balanced group of operators. The same operators will perform the same number of sliding autonomous and teleoperated runs. Additionally, the teleoperator’s complaints about the poor visual feedback demand attention. A 3D visualizer to enhance the operator’s experience is under development (a similar idea was implemented in Kortenkamp 1999). Although the visualizer will require that the autonomous fiducial recognition modules be enabled, the system will still be completely teleoperated (see Section 6.2.2). The visualizer will use the current position of objects in the world as observed by the Roving Eye to update a simulation of the world. The visualizer will allow the operator to dynamically alter his perspective without moving the Roving Eye (see Figure 28).

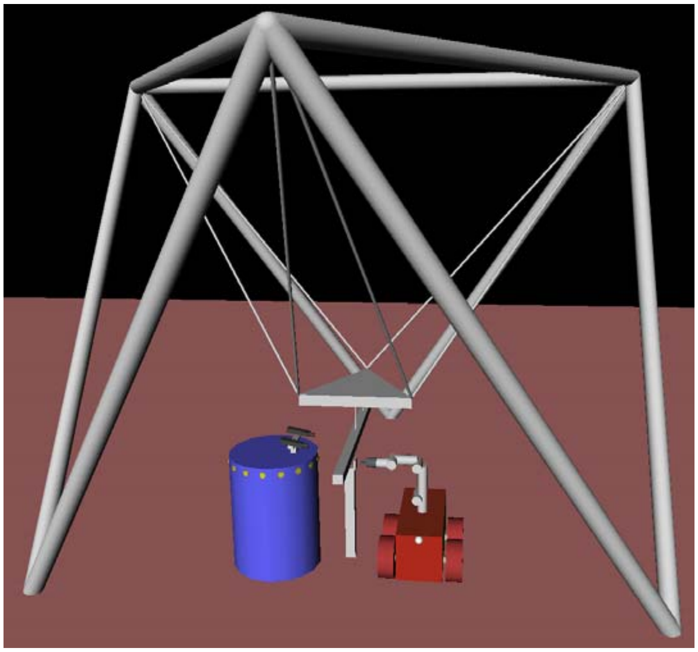


Figure 28 – A 3D environment will automatically update to reflect the positions of objects in the real world. The operator can then manipulate his perspective without moving the Roving Eye.

7.2 Extending Sliding Autonomy

There are several aspects of sliding autonomy which the experiments have not yet explored. For example, in the beam assembly task, every action within the system can be performed autonomously. The autonomous procedures may not be flawless, but the system has means to approach every task. Future work will incorporate several tasks that demand operator assistance. Such tasks will require that the autonomous system still be able to maintain some notion of success (i.e., to determine when the operator has completed his goal) without knowledge of how the task is being performed. We believe that the separation of monitors and actions supports this kind of structure, as monitors will function autonomously, while actions perform in an exclusive teleoperated mode.

Currently, switching between autonomous and teleoperated modes is performed at a task level. Monitors are used to determine when some task is complete and actions are used to progress the system towards the goal. Consider the example in Figure 29; the operator may be moving the end-effector from the first to the second waypoint. The goal of the end-effector is the third waypoint. Suppose some obstacle prevents movement toward the second waypoint. The operator has intervened and will skip the second waypoint and proceed directly to the third waypoint (path A). The autonomous system, however, is monitoring for the completion of the second waypoint and will not detect the operator's overall goal satisfaction.

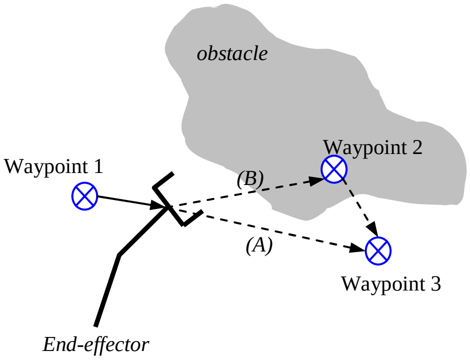


Figure 29 – A set of sequential waypoints (1,2) are used by the autonomous system to approach a goal waypoint (3). Given the current end-effector position, the autonomous system would expect the user to approach waypoint 2. The operator, however, notices the obstacle and proceeds directly to waypoint 3. The autonomous system must recognize that the user has skipped a waypoint in favor of a better route.

There are several possible ways to handle cases where the operator skips steps or performs them out of order. In the example of waypoints, monitors might be checking for both intermediate and destination waypoint completion. However, the situation becomes more complicated if the operator assumes control during the first end dock but instead dock the second end of the beam. Upon resuming, the autonomous system would need to recognize that the operator has completed some task outside his prescribed goal.

The most general case will require that the system re-observe the state of the world and re-plan for the new set of goals. In the example, if the operator completes the second end dock first, the autonomous system would have to recognize the new situation and invoke the planning layer to achieve the new goals. Observing the state of the world may be a difficult task (the system may find itself in a very ambiguous state) which may require further operator assistance. Proper implementation would also require a more complete planning layer as explored in Appendix B.

Additionally, attempts will be made to characterize tasks as best performed by the autonomous or teleoperated system. For a given task, the system will automatically determine if the task should be assigned to the operator or to the autonomous system. The judgment can be made based on the human's past performance. If, for example, the beam grasp operation is usually done more quickly and reliably by the operator, the system may elect to immediately request operator assistance before an autonomous grasp is attempted. Making such a determination would require that the system maintain some consideration of the operator's workload.

7.3 Truss Assembly

The current set of experiments for the Trestle project involves assembling a more complex version of the beam task. The Crane, Mobile Manipulator, and Roving Eye will work together to construct a square using four beams (square's edges) and four nodes (square's corners) as suggested in Figure 30. The nodes will be mounted on bases that are free to roll in any direction. This 2D freedom will begin to simulate the lack of resistive force encountered in space. To mate a node and beam, two robots will be required: one to maneuver the beam and one to steady the node.

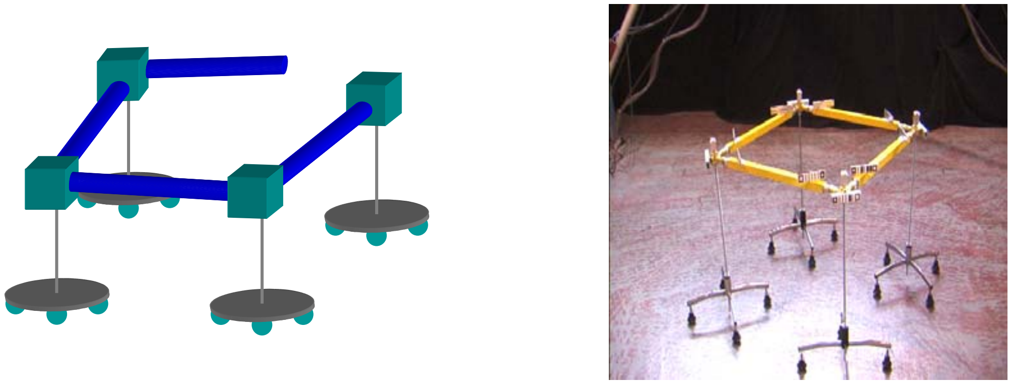


Figure 30 – The robot team assembles a free standing truss structure. The beams will be docked into nodes mounted on movable caster bases. The 2D freedom of the nodes simulates the lack of reactive forces found it space. One robot will be needed to hold the base while a second docks the beam.

The assembly process proceeds similar to that shown in Figure 32. The Crane is responsible for securing the nodes and the Mobile Manipulator servos the beam into position. Ideally, the beams would be loaded in an automatic hopper, but, for now, the beams are simply handed to the Mobile Manipulator. The Crane secures a node and the Mobile Manipulator docks the free beam into the fixed node. The Crane then releases the node and moves to grasp the node at the other end of the beam. The Crane and Mobile Manipulator then work together to dock the free node into the fixed beam. The process is then be repeated around the square.

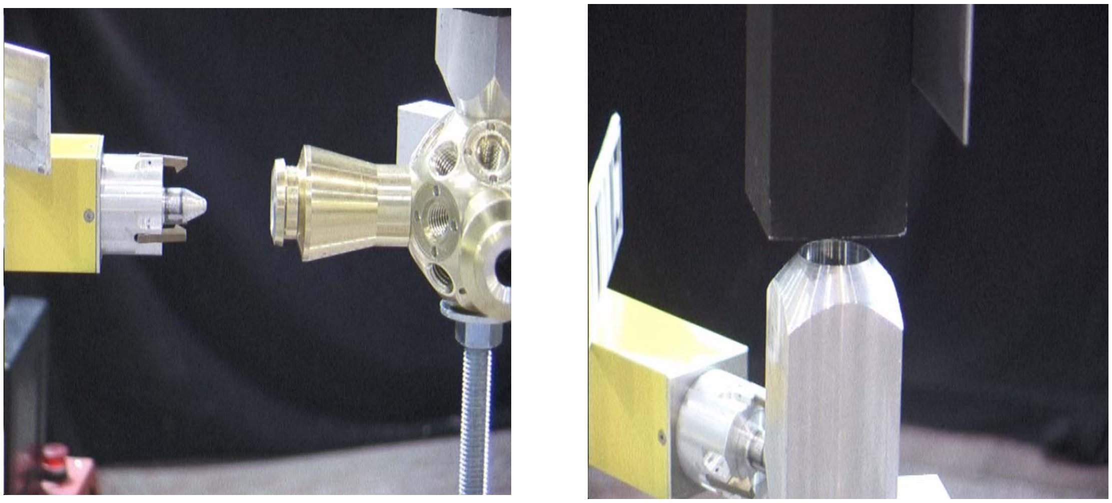


Figure 31 – (Left) The Mobile Manipulator is about to dock the beam into the node. (Right) The Robocrane aligns its end-effector autonomously and then prompts the operator for assistance.

More precisely, the entire construction of the 4-sided structure is comprised of four nearly identical single side assemblies. The assembly procedure for a single side is as follows:

1. The Robocrane is autonomously aligned to engage the node. Because the tolerance for alignment is well beyond the crane's capabilities, the operator is responsible for actually engaging the node. When this step is completed, the node is rigidly secured. See Figure 31.
2. The operator retrieves a beam using the Mobile Manipulator. At this time, the beam retrieval is too complex for the autonomous system and would require significant increases in our obstacle avoidance capabilities.
3. The system autonomously docks the beam into the node. The beam must be aligned with the receptacle's center within approximately ±0.5cm and ±5 degrees.
4. The operator moves the Roving Eye from the current node to the next node. Meanwhile, the Robocrane is autonomously moved to the next node.
5. The Robocrane autonomously aligns with the node and the operator completes the engage, as in Step 1.
6. The Mobile Manipulator autonomously docks the second end of the beam into the node. Tolerances are similar to those in Step 3. See Figure 31.
7. The Mobile Manipulator autonomously releases the beam.

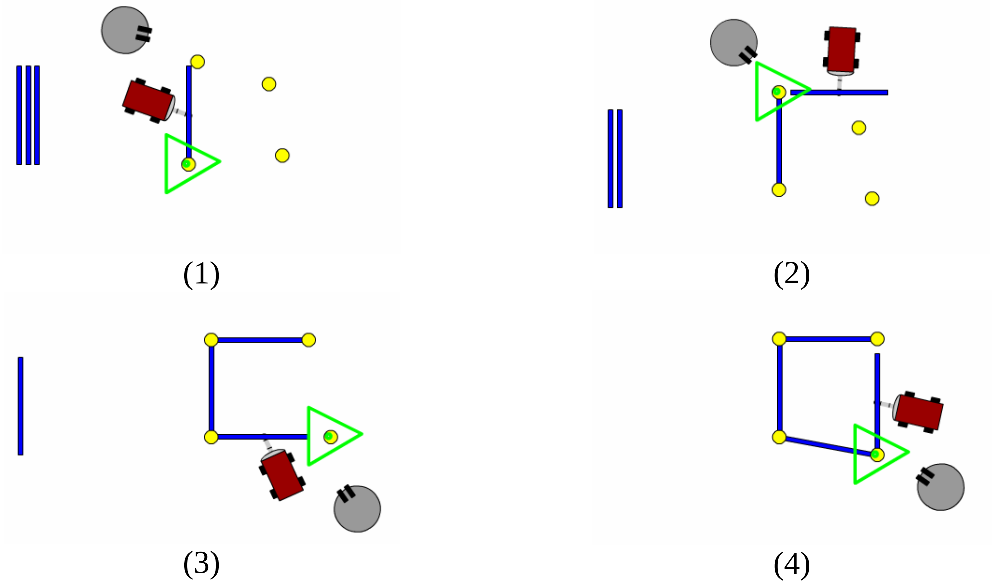


Figure 32 – The Crane (green triangle), Mobile Manipulator (red robot), and Roving Eye (gray circle) will work together to assemble a small portion of a truss structure. The assembly proceeds from left top to bottom right.

This task has significantly more steps than the previous beam assembly experiment and will likely exhibit a wider range of failures. There are many gross robot movements between dockings which are too difficult to implement autonomously and require operator assistance. The use of a 3D visualizer will be essential to help teleoperators maintain overall task cognizance. Furthermore, the extensive nature of the task tree will provide a rich environment to explore autonomous-teleoperation mode changes.

Appendix A Petri Nets

Petri Nets (PN's) were first developed by Carl Adam Petri, a German mathematician, in the early 1960s. Petri Nets can be used to visualize and model systems that include parallel, concurrent, synchronized, and exclusive processes. Then, a substantial body of supporting work makes it possible to analyze these systems. This discussion will briefly describe concepts similar to all types of PN's, and then specifically address synchronized, stochastic PN's. Much of this material is adapted from (David 1997 and Haas 2002).

A.1 Introduction

PN's provide a visualization and analysis tool for systems. Fundamentally, a PN consists of four elements: *places*, *transitions*, *arcs*, and *marks* (see Figure 33). A place may contain one or more marks. Arcs connect transitions to places (output arc) and places to transitions (input arc). Generally, when a transition *fires*, it changes the number of input marks (marks in the place(s) connected to the input arc) and output marks (marks in the place(s) connected to the output marks). An input place is a place connected to a transition via an input arc; an output place is a place connected to a transition via an output arc. *Inhibitor arcs* only travel from places to transitions and end in a bubble instead of an arrow. A transition is *enabled* when every input place contains at least one mark. When a transition *fires*, it removes one mark from every input place and adds one mark to every output place. (There are alternate techniques to describe the removal of multiple or continuous marks.) If a transition has an input inhibitor arc, then the transition cannot fire if the inhibitor place has any marks (the marking of the inhibitor place does not change when the even fires). Note that there is no conservation of marks: a transition can create and destroy marks if the number of input arcs and output arcs are not equal. A transition can fire only once it has been enabled.

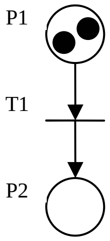


Figure 33 – A simple Petri Net (PN). There are two places, P1 and P2; a single transition, T1; and two marks in P1. T1 is enabled because its single input place, P1, contains at least one mark. When T1 fires, a mark will be removed from P1 and a mark will be added to P2.

For stochastic PN's, the time between the enabling of the transition at time t and the firing of the transition no later than at time $t + dt$, is a random variable distributed according to the exponential law. The probability that the firing occurs before $t + dt$, given that it has not occurred before t, is $P(X \le t + dt | X > t) = \mu \cdot dt$. Thus, the distribution function is $P(X \leq t) = 1 - e^{-\mu t}$. The mean time of the firing is $\frac{1}{\mu}$ with a variance of $\frac{1}{\mu^2}$. Combined with additional analysis tools and based on this mean firing rate, performance of a PN described system can now be considered.

In Figure 33, T1 is 2-enabled because T1 could be fired twice consecutively. In other words, a system is 1-enabled (or simply enabled), when each of its input places contains at least one mark. If each of a transition's input places contains q marks, then the transition is said to be *q-enabled*. In a *synchronized* PN, a q-enabled transition fires only when its associated event occurs (in the case of a stochastic PN, this event occurs a random time after enabling). In Figure 33, the event associated with T1 would have to occur twice before two marks appear in place P2.

The *state* of a PN is a vector describing the number of marks in a place, M(P). For Figure 33, the state vector is [M(P1), M(P2)] and is [2,0]. If T1 fired once, the state would be [1,1]; if T1 fired another time the state would be [0,2]. Every assignment of marks to places, or *marking*, corresponds to a state of the system. A transition may change the marking of the system and cause a *state transition*.

A simple algorithm can be used to interpret a synchronized, stochastic PN (adapted from David 1992):

1. Set the system to the initial state at $t = 0$.
2. Let $X = \{T_1, T_2, \cdots, T_j, \cdots\}$ be the set of enabled transitions.
3. For each transition T_j of set X, chose a firing time d_j based on the exponential law and μ_j (μ_j is the mean firing rate associated with T_j). If T_j is q-enabled, chose firing time d_j based on the mean firing frequency $q \cdot \mu_j$.
4. Let $d_m = \min(d_1, d_2, \cdots, d_j, \cdots)$ and T_m be the associated transition.
5. Fire transition T_m and update the system's state.
6. Set $t = t + d_m$.
7. Repeat at step 2.

Step 3 may seem counter intuitive in that q-enabled transitions have a mean frequency q-times greater than that of a 1-enabled transition. Remember, however, that μ_j corresponds to the frequency at which the event associated with the transition will occur and cause the transition to fire. Thus, when a transition is q-enabled, it is more likely that the transition will fire because the event can occur for any of the q marks available.

No place may have fewer than zero marks. As a result, a *conflict* may occur if two transitions were to fire and attempt to remove the same mark from a single place. In Figure 34, the current state of the system is [P1,P2,P3,P4]=[1,1,0,0]. If T1 fires first, the state will be [0,0,1,0] and T2 will not be able to fire. If T2 fires first, the state will be

[0,0,0,1] and T1 will not be able to fire. Thus, T1 and T2 are said to conflict because they compete for marks. According to the algorithm, the conflict is resolved based on the firing frequencies of the transitions: the first transition chosen to fire consumes the marks and may leave conflicting transitions disabled.

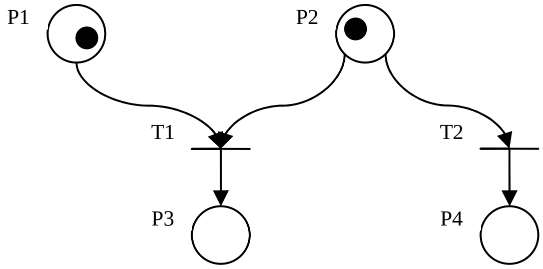
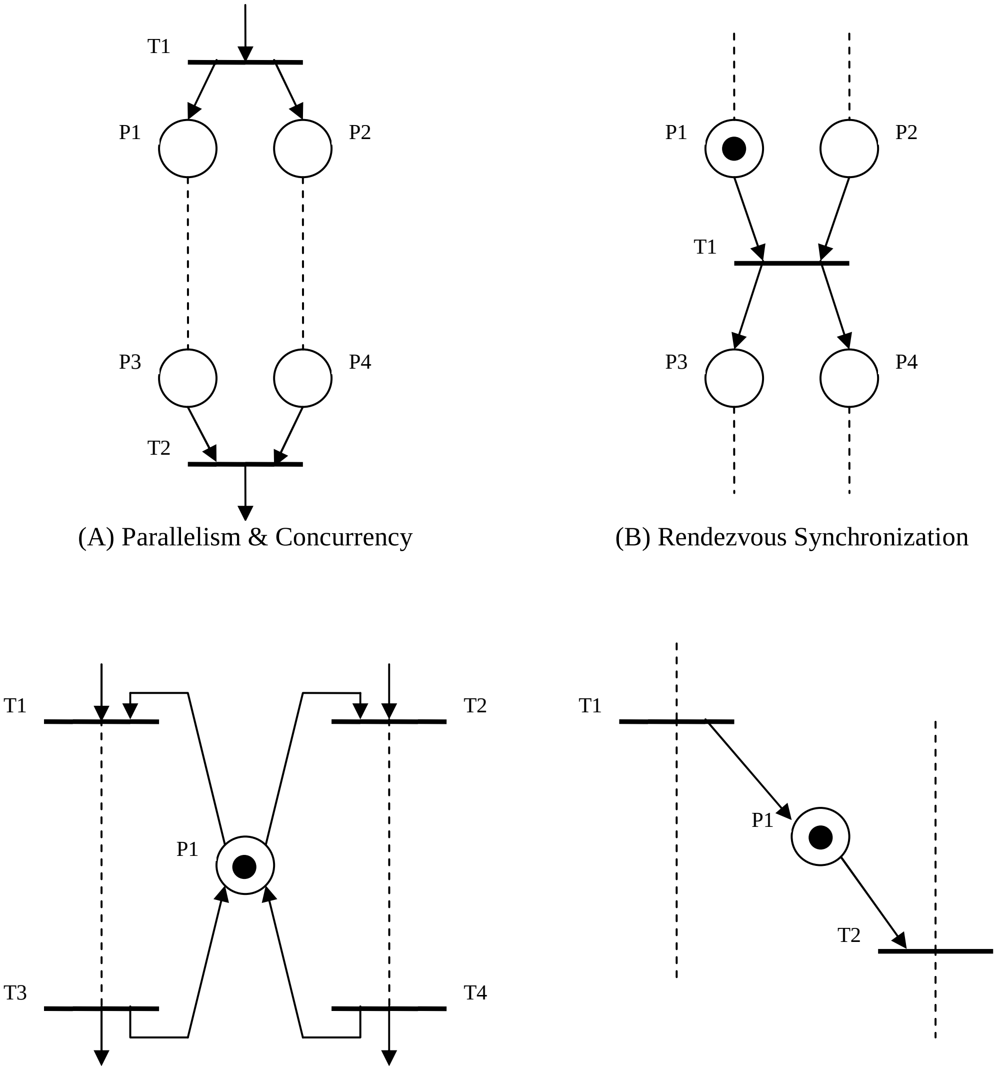


Figure 34 – T1 and T2 are both enabled and have mean firing frequencies μ_1 and μ_2, respectively. T1 and T2 both are enabled based on the mark in P2 and conflict.

Figure 35 shows a sampling of the wide variety of system functionalities that PN's can demonstrate. Concurrency can be represented by branching flow. Synchronization can be represented by forcing two branches to recombine and serve as a transition's input. Resource sharing allows a PN to reflect limited constraints.



(A) Parallelism & Concurrency

(B) Rendezvous Synchronization

(C) Resource Sharing

(D) Semaphore Synchronization

Figure 35 – A PN allows visualization of many different system interactions.

A.2 An Example

A PN may be analyzed by first examining every possible marking of the system and determining the frequency with which the system transitions between these states. Assuming the state space is finite, the analysis is then analogous to that of a continuous time discrete state space Markov process. For every marking, consider every transition that is enabled. Fire each transition from the original marking and generate new markings. Repeat this process until all markings have been explored and a complete state diagram will remain.

For example, consider a two step processing system in Figure 36. The process consists of two repeating steps P1/T1 and P2/T2. Marks in P1 indicate objects that are undergoing some manufacturing process, perhaps a drying process. Marks in P2 indicate objects that are undergoing a different process, perhaps a painting activity. The number of painting robots, however, is limited to the marks in P3 – in this case, three. At any point, no more than 3 objects can undergo painting at the same time (thus, P3 adds a resource constraint to the system). There is no such limit, however, on the number of objects that can dry simultaneously. The transition T1 occurs when an object finishes drying and is ready to be painted again. The mean frequency, based on an exponential law, with which an object completes drying, depends on the number of objects that are currently drying. Similarly, the rate at which transition T2 fires and indicates the completion of a painting activity depends on the number of objects currently being painted.

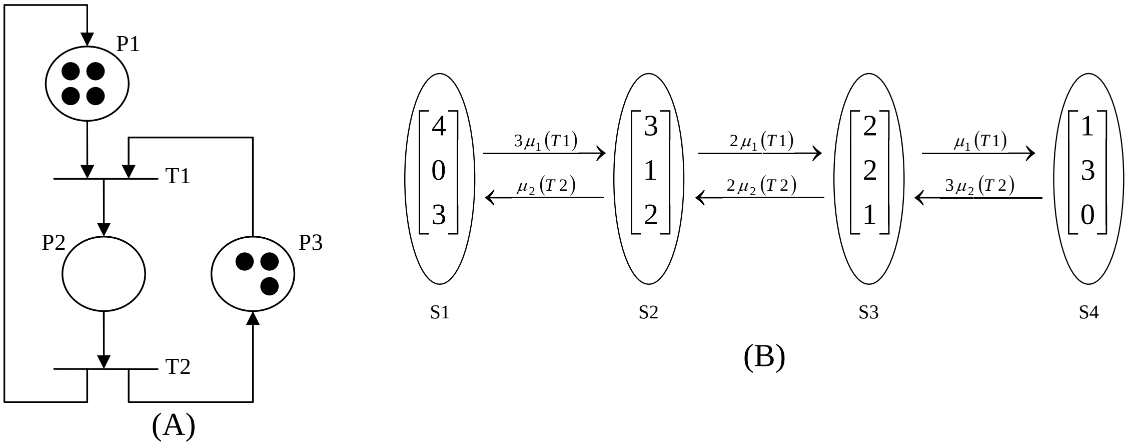

P1
T1
P2
P3
T2
(A)
$\begin{bmatrix}4\\0\\3\end{bmatrix}$
$\begin{bmatrix}3\\1\\2\end{bmatrix}$
$\begin{bmatrix}2\\2\\1\end{bmatrix}$
$\begin{bmatrix}1\\3\\0\end{bmatrix}$
$3\mu_1(T1)$
$\mu_2(T2)$
$2\mu_1(T1)$
$2\mu_2(T2)$
$\mu_1(T1)$
$3\mu_2(T2)$
S1
S2
S3
S4
(B)

Figure 36 – A sample PN. A repetitive painting and drying process can easily be analyzed with a PN. Example adapted from (David 1997).

In Figure 36B, S1-S4 illustrate the four possible states of the system. The transitions between states correspond to transition firings. Additionally, based on a q-enabled weight factor, each transition occurs with some exponential rate. For example, from the initial state S1, the system will transition to S2 with a mean firing time of $3\mu_1$ since transition T1 is 3-enabled (three is the minimum number of marks of P1 and P3).

A markovian process matrix can now be constructed. Each element in the matrix, a_{ij} is the mean transition rate from state *i* to state *j*. The diagonal elements are negative of the mean transition rates away from a state. For the example,

$$A = \begin{bmatrix} -3\mu_1 & 3\mu_1 & 0 & 0 \\ \mu_2 & -2\mu_1 - \mu_2 & 2\mu_1 & 0 \\ 0 & 2\mu_2 & -2\mu_2 - \mu_1 & \mu_1 \\ 0 & 0 & 3\mu_2 & -3\mu_2 \end{bmatrix}.$$

The probability of being in a particular state can be found by solving $PA = 0$ where P is a vector of the state probabilities. Furthermore, the probabilities must sum to one. The mean firing frequency of each transition can be found by summing the products of the transition rates by the state probabilities. The mean number of marks in a place can be determined by summing the products of the number of markings in each state by the probability of being in that state. Finally, the mean dwelling time can be determined by applying Little’s formula (divide the mean number of marks by the mean firing frequencies of the output arcs).

In many ways PN’s are similar to state automata. Both are graphical representations with an accompanying body of tools which can aid system analysis. Certain types of PN’s and state automata can model non-determinism and it is possible to convert between certain types of PN’s and state space Markov chains. PN’s, however, explicitly represent enabled transitions and state automata explicitly represent a system’s states. Of particular interest here, PN’s also provide excellent models for several complicated system interactions mentioned earlier.

Appendix B The Planning Layer

The Distributed Robot Architecture (DiRA) and Trestle projects described in this thesis both use a multi-layered robot control architecture. The layers include a high level planning layer, a middle level executive layer, and a hardware-centric lower behavior layer. Currently, the planning layer is a fixed list and cannot be changed. The DiRA project, for example, implicitly assumes a plan consisting of {(dock first end), (turn to second end), (dock second end)}.

A fixed plan performed reasonably well for the purely autonomous system. Much of the recovery and exception handling – the kinds of events that might require re-planning – were handled at the executive layer. However, as we begin to consider sliding autonomy, we would like to have the operator be able to change the plan during plan execution. During the autonomous trials for the DiRA project, several failures were due to the first end coming undocked while docking the second end. The autonomous system had no way to determine such a failure and the operator had no means to force the system to repeat the first end dock. In such cases, and others, the system needs to be able to re-plan given the current state to achieve the original and/or operator-modified goals.

The following discussion explores some of the current planners for deterministic (each action has a certain result) and non-deterministic (each action has results which occur with some probabilities) domains. For the purposes of this discussion, we will consider a slightly simplified version of the Trestle project to construct a truss frame. The frame will have four equal length sides, referred to as beams, and four corners, referred to as nodes. In this version of the experiment, the nodes are suspended from the ceiling, in position, to approximate the lack of reactive forces that might be found in space. The beams are located in a stockpile and can only be carried from one location to another by Robocrane. To mate a beam and node, Bullwinkle must be holding that side of the beam and another robot, Trolley, must be holding the node. In all, eight mating operations are needed to construct the frame.

B.1 The Planners

In general, a wide variety of planners might be appropriate for the planning layer. Partial-order planning was a potential candidate, but preliminary tests revealed that the available implementations were simply too slow to develop a plan for the simplified Trestle project.

The Fast-Forward (FF) is a heuristic-search planner which develops heuristics while planning. First, the GraphPlan planner (Blum 1995) is used to develop a relaxed plan which ignores negative interactions between domain actions (i.e., ignores condition delete lists). This relaxed plan can then be used to estimate the distance to a goal and prioritize successor actions for search. Enforced hill climbing is then used to explore those suggested successor actions. FF is neither optimal nor complete (Hoffman 2001).

As a planning algorithm for the planning layer, the FF algorithm would be called to develop a complete plan and then, if necessary, called again to develop completely new plans from scratch if exception conditions existed.

A Binary Decision Diagram (BDD) is a tree representing a Boolean function (Cimatti 1998). Each node in the tree has two leaves, corresponding to true and false evaluations. All paths in the tree terminate at either a true or false evaluation. Using an ordered BDD (OBDD), the Universal Multi-agent OBDD-based Planner (UMOP) represents the state transitions of the domain and attempts to develop a plan. A unique characteristic of UMOP is that it allows actions (and the environment) to have unpredictable characteristics. By properly designing a domain, an action can have several possible results. This presents an interesting opportunity for the Trestle domain as much of the motivation for sliding autonomy is to handle these unexpected events. UMOP can develop several kinds of plans. A strong plan is a plan that will guarantee goals are satisfied; a strong cyclic plan is a plan that guarantees goals are satisfied only if some set of non-deterministic actions produces a desired result at least once; an optimistic plan is a plan that guarantees that the goals are met only if some set of non-deterministic actions produces a desired result at a particular time. (Jensen 2000)

The deterministic domain was written in PDDL and planned with FF v2.3. The non-deterministic domain was written in NADL and planned with UMOP as a BDD representation.

B.2 The Deterministic Domain

The deterministic representation of the experiment has two iterations. In the first, each robot could grab any beam or node and move to any location. FF produced a sound plan for the frame's construction, but the plan was far from optimal. In more than a few cases, robots grasped and then immediately released objects and traveled to locations needlessly. Of course, FF is not an optimal planner. Nonetheless, because the planning at this stage is at such a high level, any unnecessary tasks can have substantial costs.

The next iteration attempted to structure the way in which FF could choose a solution. From some perspectives, this kind of structuring is a bit of a "cheat," essentially helping the planner along to a better solution. From other perspectives, however, the guidelines represent the domain more accurately, by expressing the high objection to needless steps. The restrictions imposed on construction are that Bullwinkle and Trolley may only move to adjacent locations. For example, Bullwinkle can only mate beams in the order 1,2,3,4 and Trolley can only mate nodes in the order A,B,C,D. This restriction, not necessarily essential to the domain, forced FF to choose plans that built the frame in an organized manner and reduced the number of movements and grasps. The restrictions also forced FF to complete each beam or node before beginning another beam or node sequence.

For example, the system had to finish docking both sides of beam 1 before proceeding to beam 2. The one exception is node A, the first node, which must be released without being completely assembled (the frame is assembled in a sequential, clockwise fashion).

Again, this restriction prevented the system from roaming from one location to another and completing only part of a job at a time.

In the end, the deterministic plan presents a straight-forward, efficient means of constructing the frame. The resulting domain also shows that the assembly is nearly a repeated set of actions. The actions secure a node on the left, mate the beam on the left, secure the node on the right, and mate the beam on the right. Only the first node, node A, needs special attention in the trolley-grasp function, because it must be grasped twice even in the most efficient implementation.

B.3 The Non-Deterministic Domain

For the non-deterministic domain, an MDP representation was initially anticipated. However, given the number of states for the frame assembly and only a vague guess for probabilities, it became clear that this was beyond the time frame of this project. Instead, the domain was represented in NADL and used Jensen's UMOP v1.2 planner to analyze the domain. (To facilitate the work, UMOP was changed and recompiled so that the initial conditions were defaulted by the initial problem state. This new compilation was tested for all the following cases against the original build of UMOP to verify that the changes did not alter the results.) UMOP developed a cyclic strong universal plan which was then used to see how the system handled different kinds of failures.

The development of the NADL description also had two iterations. The first iteration was a naïve, straightforward conversion of the PDDL description. Using a quick program, the PDDL functions were converted to NADL functions by exploding the parameters. When a PDDL function had one parameter with two values, the NADL description had two functions, each representing a bound version of the original PDDL function. Clearly, this representation was not efficient (due to the large number of functions) and the NADL description exceeded 100Kb in size. UMOP was only able to develop plans in a reasonable amount of time (less than an hour) for relatively simple goals (goals requiring few steps). Goals, such as mating even a single beam, proved untimely. The function space could be quickly reduced, however, by eliminating bindings that were unnecessary or which contradicted preconditions. Even with this minimization, the description still performed poorly and was not able to find plans for a complete frame assembly in a reasonable amount of time.

The second iteration proved to be more efficient. The main difference was the use of value variables to represent locations, objects held, etc., rather than just Boolean variables. Additionally, the Trolley agent was eliminated and the nodes were always assumed to be grasped. (The Trolley agent added to the accuracy of the domain, but did not add significantly to the discussion of the output.) Like the deterministic domain, the goals of the non-deterministic domain were to construct the frame and leave both robots' manipulators empty. Beams were placed in a stockpile location (location 0) and had to be brought into position by the Robocrane and docked by Bullwinkle.

Despite the presence of the noop operator, Bullwinkle tended to wander aimlessly while waiting for the crane. Again, the solution was to force the planner to move Bullwinkle a minimum number of times (by limiting Bullwinkle to move in sequence around the frame).

For the NADL description, non-determinism came in two forms. The first was the mating procedure. By design, mating will be the most difficult aspect of the assembly. As a result, the dock operators may or may not succeed, leaving that side of the beam either docked or undocked. This first ambiguity makes the plan only cyclic strong; the only way to recover is to attempt another dock operation, which may also fail. During one run, for example, the attempt to dock the first beam had to be done three times before it succeeds (normally two docks are required for each beam, one for each side).

The second form of non-determinism was the environment action that caused Bullwinkle's grasp to slip. When this operator occurred, Bullwinkle lost its grip on the beam and had to re-grasp. This potential error also made the plan only cyclic strong.

The final plan succeeded in constructing the beam after 47 steps. It was curious to note that, at several points, the planner executed noop operators for both robot and environment agents. Although not incorrect from the domain definition standpoint, the execution of these noops was certainly not necessary.

B.4 Conclusions

The deterministic plan provides a clear description of the high level tasks that will have to be accomplished by the individual agents. The non-deterministic BDD representation gives a sense for how the system could be expected to autonomously recover from errors. Beyond this Appendix, the next step will be to more closely simulate the experiment, possibly using a MDP representation where operations exist which request operator action. These operations will have a high probability of success but they will penalize the system.

Appendix C Crane Control

The Robocrane, or simply the crane, is a 6-DOF Inverted Stewart Platform (ISP). The end-effector of the crane is an equilateral triangle (approximately one meter on an edge) which is suspended from base equilateral triangle (approximately three meters on an edge). The cables which support the end-effector can be changed in length (prismatic joints) and are attached to the base and end-effector with revolute (spherical) joints. The crane is, thus, known as a revolute-revolute-prismatic actuator.

For the Distributed Robot Architecture (DiRA) project, the crane was controllable through relative and absolute position commands only. This methodology had two primary shortcomings. First, the movement of the end-effector between two points was not necessarily linear. Given its current position and a desired end-effector configuration, the crane would calculate the necessary joint positions. Because the joints had different distances to travel and because the speed of the joint actuation was fixed, the end-effector would move through a non-linear path as it approached the goal. Furthermore, because not every joint configuration is valid, the crane could easily pass through invalid configurations while moving between valid positions. An invalid position is one in which a joint must take an unachievable value. As a result, a path planner was employed to navigate around invalid positions resulting in discontinuous end-effector motion. Second, efforts toward implementing sliding autonomy required that each robot have an operator controllable mode. For the crane, the natural control device was a 6-DOF "space mouse" allowing the user to navigate the crane with all freedom. Position control was inappropriate for mouse-based control because of the non-linearity of the system and because the position control was uninterruptible. Uninterruptible motion would be unnatural for the operator to use.

Due to the deficiencies of the position controller, interruptible velocity control of the crane is needed. Velocity control would allow the operator to 'fly' the crane naturally with the space mouse and produce linear end-effector movements. The following analysis reveals the necessary calculations to determine joint velocities based on a desired end-effector velocity.

C.1 The Inverse Jacobian

The ISP is similar to the Stewart Platform (SP) shown in Figure 37, except that the end-effector is below the base rather than above the base. For analysis purposes, the two structures are the same. The upper B triangle is the end-effector; the lower A triangle is the base triangle. Though drawn as triangles, the following analysis will support any general hexagon. Strictly speaking, the crane is actually hexagonal as the joints do not meet at exactly the same points (and so the "triangle" actually has six sides). Point C is the rotational center of the end-effector and point O is the center of the base. Each of the six joints connects a point on the A triangle to a point on the base triangle. The vector $\vec{q}_i$

describes the ith joint; the vector $\hat{q}_i$ is the unit vector along the ith joint; the length of the ith joint can then be determined by evaluating $\left\|\vec{q}_i\right\| = \left\|\overrightarrow{AB_i}\right\| = d_i$.

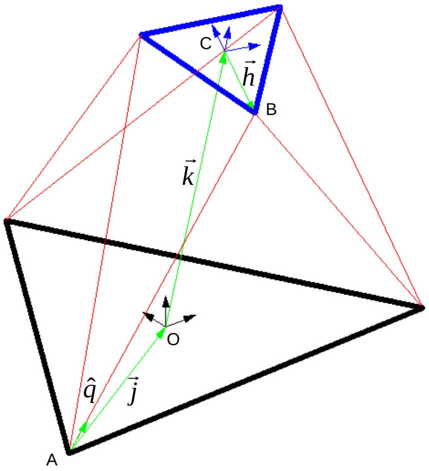


Figure 37 – A geometric representation of the ISP.

The following derivation of the Inverse Jacobian follows closely with that described by (Merlet, 00). Given some end effector velocity, $\dot{X}$, it is desired to find the needed joint velocities, $\dot{Q}$, related by $\dot{Q} = J^{-1}\dot{X}$. The inverse Jacobian, J^{-1}, will be a 6x6 matrix (there are 6-DOF in X and 6 actuated joints in Q). Let R_C^O be the rotation of the C-frame relative to the O-frame and h_i^C be the vector $\overrightarrow{CB_i}$ expressed in the C-frame. Considering the ith joint (and drop the i's), it is clear that:

$$\vec{q} = \vec{j} + \vec{k} + \overrightarrow{h^O} = \vec{j} + \vec{k} + R_C^O\overrightarrow{h^C}$$

$$\left\|\vec{q}\right\|^2 = \vec{q}\cdot\vec{q} = d^2$$

$$d^2 = \left\|\vec{j}\right\|^2 + \left\|\vec{k}\right\|^2 + \left\|\overrightarrow{h^C}\right\|^2 + 2\vec{j}\cdot\vec{k} + 2R_C^O\overrightarrow{h^C}\cdot\vec{k} + 2\vec{j}\cdot R_C^O\overrightarrow{h^C}$$

Note that $R_C^O\overrightarrow{h^C}\cdot R_C^O\overrightarrow{h^C} = \left\|\overrightarrow{h^C}\right\|^2$ since $\left\|R_C^O\right\| = 1$.

Now calculate the derivative of one of the ith joint with respect to time:

$$2d\dot{d} = 2\vec{j}\cdot\underbrace{\dot{\vec{j}}}_{=0} + 2\vec{k}\cdot\dot{\vec{k}} + 2\overrightarrow{h^C}\cdot\underbrace{\overrightarrow{\dot{h}^C}}_{=0} + 2\vec{j}\cdot\dot{\vec{k}} + 2\underbrace{\dot{\vec{j}}}_{=0}\cdot\vec{k} + 2\overrightarrow{\dot{h}^O}\cdot(\vec{k}+\vec{j}) + 2\overrightarrow{h^O}\cdot(\dot{\vec{k}} + \underbrace{\dot{\vec{j}}}_{=0})$$

$$2d\dot{d} = 2\vec{k}\cdot\dot{\vec{k}} + 2\vec{j}\cdot\dot{\vec{k}} + 2\overrightarrow{\dot{h}^O}\cdot(\vec{k}+\vec{j}) + 2\overrightarrow{h^O}\cdot\dot{\vec{k}}$$

$$d\dot{d} = \dot{\vec{k}}\cdot\left(\vec{k} + \vec{j} + \overrightarrow{h^O}\right) + \overrightarrow{\dot{h}^O}\cdot(\vec{k}+\vec{j})$$

$$d\dot{d} = \dot{k}\cdot\vec{q} + \overrightarrow{\dot{h}^O}\cdot\overrightarrow{AC}$$

Let $\dot{X} = \begin{bmatrix} V \\ \Omega \end{bmatrix}$ where V and Ω correspond to the translation and rotational velocities of the center of the end-effector relative to the center of the base, respectively. Note that $\dot{\vec{h}}^{O} = \overrightarrow{BC} \times \Omega$.

$$\dot{d} = V \cdot \frac{\vec{q}}{d} + \dot{\vec{h}}^{O} \cdot \frac{\overrightarrow{AC}}{d}$$
$$\dot{d} = V \cdot \hat{q} + \left(\overrightarrow{BC} \times \Omega\right) \cdot \frac{\overrightarrow{AC}}{d}$$
$$\dot{d} = V \cdot \hat{q} + \left(\overrightarrow{BC} \times \Omega\right) \cdot \frac{\overrightarrow{AC}}{d}$$
$$\dot{d} = V \cdot \hat{q} + \Omega \cdot \left(\frac{\overrightarrow{AC}}{d} \times \overrightarrow{BC}\right)$$
$$\dot{d} = V \cdot \hat{q} + \Omega \cdot \left(\frac{\overrightarrow{AB} - \overrightarrow{BC}}{d} \times \overrightarrow{BC}\right)$$
$$\dot{d} = V \cdot \hat{q} + \Omega \cdot \left(\hat{q} \times \overrightarrow{BC}\right)$$

Thus, the ith row of the Inverse Jacobian can be built as follows:
$J_i^{-1} = \begin{bmatrix} \hat{q}_i^{T} & \left(\hat{q}_i \times \overrightarrow{BC_i}\right)^{T} \end{bmatrix}$.

The Inverse Jacobian can now be constructed in closed form, given the position of the ith attachment point relative to the center of the end-effector, $\overrightarrow{BC_i}$, and the direction of the ith joint, $\hat{q}_i$. The Inverse Jacobian is relatively simple given that the velocity of each joint is independent of the current positions of all other joints. Calculating $\hat{q}_i$ will require the calculation of the forward kinematics and determination of the position of the end-effector given the current joint positions.

C.2 The Forward Kinematics

The goal of forward kinematics is to calculate the 6-DOF position of the crane's end-effector given the six joint positions, defined as $d_1, \cdots, d_6$ in the previous section. The forward kinematics for the ISP are identical to those of the SP. For a given set of joint values, there can be many realizable positions of the end-effector (the exact number depends on the geometric configuration and joint values). For an ISP, it may be possible to eliminate solutions based on a minimal energy constraint: Gravity will tend to force the end-effector towards a least potential-energy solution. However, since such a technique still may not suggest a unique solution, the current position of the end-effector is determined by choosing the configuration nearest the most previous configuration. Because the forward kinematics will be run continuously in a velocity control loop and the end-effector will have moved only a small amount, this is a reasonable strategy.

The following discussion follows closely with that by (Merlet, 00) and (Griffis, 89) and will first define variables and reference frames. Second, the equations to solve the forward kinematics will be presented. Third, these equations will be separated and linearized. Fourth, an iterative technique for solving the equations will be discussed.

C.2.1 Definitions

In Figure 38, the A triangle is again the base triangle and the B triangle is again the end-effector triangle. To simplify the forward kinematics, both the base and end-effector are assumed to be perfect equilateral triangles with edges of length a and b, respectively.

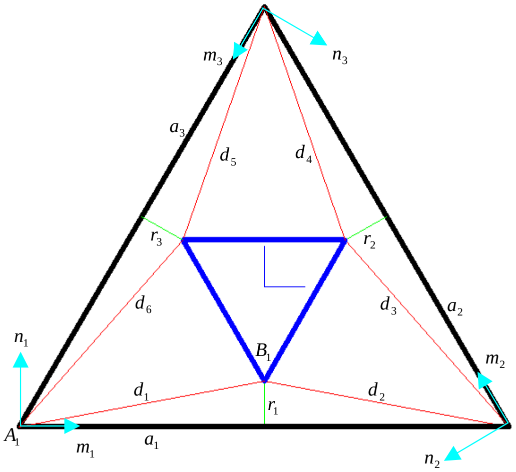


Figure 38 – An overhead view of the ISP. The center (blue) triangle is the end-effector; the outer (black) triangle is the base.

At each corner of the base triangle, a set of axes where the m_i axis points along the ith side of the triangle as shown is described. The z-vector is out of the page. The distance along m_i to the intersection with r_i is a_i. For example, $a_1 = \dfrac{d_1^2 - d_2^2 + a^2}{2a}$ by the cosine-angle theorem and a_2, a_3 are defined similarly. The length of the r vectors is easily found using the right triangle a-r-d. For example, $r_1 = \sqrt{d_1^2 - a_1^2}$.

Not shown in the above image are the angles, $\theta_1, \theta_2, \theta_3$, which describe the elevation of r_1 out of the m-n plane. This angle is also the elevation of the triangle A1-A2-B1 out of the m-n plane and these three angles represent the space by which the position of the end-effector can be uniquely specified. The method presented here attempts to find these angles.

C.2.2 Forward Kinematic Equations

Given some joint positions, $d_1, \cdots, d_6$, the goal is to determine the locations of B_1, B_2, B_3. Note that for a pair of joints, the corresponding B vertex can be anywhere on a circle swept out by the corresponding r vector. As suggested earlier, the goal is to constrain the angle of the r vector. The individual B vertices are constrained to lie on the equilateral triangle and must be a distance of b from each other.

The constraints relating the triangle A1-B1-B3 are:

$$\vec{r}_1 = \vec{c}_1 + r_1 c_1 \hat{n}_1 + r_1 s_1 \hat{z}$$

$$\vec{r}_3 = \vec{c}_3 + r_3 c_3 \hat{n}_3 + r_3 s_3 \hat{z}$$

$$b = \left\| \vec{r}_1 - \vec{r}_3 \right\|$$

$$b^2 = \left(\vec{r}_1 - \vec{r}_3\right) \cdot \left(\vec{r}_1 - \vec{r}_3\right) \Rightarrow e_{13} c_1 c_3 + e_{23} s_1 s_3 + e_{33} c_3 + e_{43} c_1 + e_{53} = 0$$

where

$$e_{13} = r_3 r_1 \hat{n}_3^T \hat{n}_1$$

$$e_{23} = r_3 r_1$$

$$e_{33} = r_3 \hat{n}_3^T \left[\vec{a}_1 - \vec{a}_3\right]$$

$$e_{43} = -r_1 \hat{n}_1^T \left[\vec{a}_1 - \vec{a}_3\right]$$

$$e_{53} = \frac{b^2 - r_3^2 - r_1^2 - \left\| \vec{a}_1 - \vec{a}_3 \right\|^2}{2}$$

The construction for the remaining two analogous triangles is similar and the three resulting equations are:

$$e_{11} c_1 c_2 + e_{21} s_1 s_2 + e_{31} c_1 + e_{41} c_2 + e_{51} = 0$$

$$e_{12} c_2 c_3 + e_{22} s_2 s_3 + e_{32} c_2 + e_{42} c_3 + e_{52} = 0$$

$$e_{13} c_1 c_3 + e_{23} s_1 s_3 + e_{33} c_3 + e_{43} c_1 + e_{53} = 0$$

From these three equations, it is possible to solve for the three unknowns $\theta_1, \theta_2, \theta_3$. This equation, however, is non-linear in the space of the three unknowns and needs to be linearlized.

C.2.3 Separation & Linearization

Let $t_i = \tan\left(\theta_i / 2\right)$, apply the substitutions $c_i = \frac{1 - t_i^2}{1 + t_i^2}$, $s_i = \frac{2t_i}{1 + t_i^2}$, and three new equations develop for i=1,2,3:

$$\tilde{e}_{1i} t_i^2 t_{i+1}^2 + \tilde{e}_{2i} t_i^2 + \tilde{e}_{3i} t_{i+1}^2 + \tilde{e}_{4i} t_i t_{i+1} + \tilde{e}_{5i} = 0$$

where

$$\tilde{e}_{1i} = e_{1i} - e_{3i} - e_{4i} + e_{5i}$$

$$\tilde{e}_{2i} = -e_{1i} - e_{3i} + e_{4i} + e_{5i}$$

$$\tilde{e}_{3i} = -e_{1i} + e_{3i} - e_{4i} + e_{5i}$$

$$\tilde{e}_{4i} = 4 e_{2i}$$

$$\tilde{e}_{5i} = e_{1i} + e_{3i} + e_{4i} + e_{5i}$$

Next, let:

$$A = \tilde{e}_{12}t_2^2 + \tilde{e}_{32} \qquad D = \tilde{e}_{13}t_1^2 + \tilde{e}_{23} \qquad G = \tilde{e}_{11}t_1^2 + \tilde{e}_{31}$$
$$B = \tilde{e}_{42}t_2 \qquad E = \tilde{e}_{43}t_1 \qquad H = \tilde{e}_{41}t_1$$
$$C = \tilde{e}_{22}t_2^2 + \tilde{e}_{52} \qquad F = \tilde{e}_{33}t_1^2 + \tilde{e}_{53} \qquad I = \tilde{e}_{21}t_1^2 + \tilde{e}_{51}$$

and write two new equations:

$$At_3^2 + Bt_3 + C = 0$$
$$Dt_3^2 + Et_3 + F = 0$$

Using the eliminant method (Griffis, 89), it can be concluded that:

$$\begin{vmatrix} AE - BD & AF - CD \\ CD - AF & CE - BF \end{vmatrix} = 0\,.$$

It is interesting to note that this determinant will be in terms of only t_2 and of the form $Lt_2^4 + Mt_2^3 + Nt_2^2 + Pt_2 + Q = 0$. When combined with $Gt_2^2 + Ht_2 + I = 0$, the eliminant method can again be applied to conclude that:

$$\begin{vmatrix} HL - GM & IL - GN & -GP & -GQ \\ GN - LI & GP + HN - MI & GQ + HP & HQ \\ G & H & I & 0 \\ 0 & G & H & I \end{vmatrix} = 0\,.$$

The resulting equation will be a polynomial in terms of only t_1. The polynomial will be 16th order in t_1^2 (i.e., truly only 8th order). Once t_1 has been found, it can be substituted back to find the remaining two angles.

C.2.4 Solving

Since the equation for t_1 is simply a polynomial, the Newton-Raphson root finding algorithm can be used. Because there may be multiple roots, the previous value of t_1 serves as an initial guess.

C.3 Results

The algorithm described was implemented on the crane and velocity control was completed. Velocity control can be interrupted and the crane moves in a nearly linear path. Small deviations are believed to be related to assumptions that the base and end-effector are perfect equilateral triangles and small errors in measured lengths.

Appendix D The Arcsecond System

This appendix investigates a potential solution for autonomous calibration of an Arcsecond, Inc., laser-based positioning system. The Arcsecond system may be a more accurate replacement for the Roving Eye. The Arcsecond system uses four laser transmitters, each projecting two tilted laser half-planes and a synchronizing infrared pulse. A receiver, detecting the lasers and IR pulses, can determine its relative horizontal and vertical angles from the transmitter. With data from at least two transmitters and their relative positions, a receiver can determine its position. Currently, the system must be manually calibrated to determine relative transmitter positions by sampling positional data at several different fixed distances. The system then uses a modified form of bundle adjustment to estimate the transmitter locations. This report demonstrates a computer model of the Arcsecond system and one possible solution enabling the use of sampled positional data even if the distances are not fixed or exact.

D.1 Motivation & Background

During advanced construction, it is necessary to know the position of robots, manipulators, materials, tools, etc. The Arcsecond system provides a means where, as long as line-of-sight is available, positions can be relatively determined with high accuracy. While the Arcsecond system may not be feasible in actual space applications, it provides a convenient positioning system for demonstration purposes. For its advantages, however, the Arcsecond system must be manually calibrated. This calibration, described in more detail shortly, requires that sample points be acquired. Some of these sample points must be at exact positions. In the demonstration, these points will have to be sampled with one or more receivers mounted on a robot. However, the robot will only be able to move the receiver to the sample locations with some uncertainty: the exact positions currently required by the system will not be available. A system that can estimate transmitter location with a less rigid set of sample points is needed.

A typical Arcsecond system setup is shown in Figure 39. The four transmitters are roughly placed at the four corners of a rectangle. While this is the arrangement Arcsecond suggests as the best, it is not essential that the transmitters be in any particular arrangement or at any exact distance from any other transmitter. (There are a few transmitter placement restrictions, but they are not considered here.) The receiver can be positioned where it has line-of-sight to all the transmitters, including spaces outside the enclosed rectangle. The receiver reports its set of measured angles via RS232 or wireless connection to a PC running the Arcsecond software.

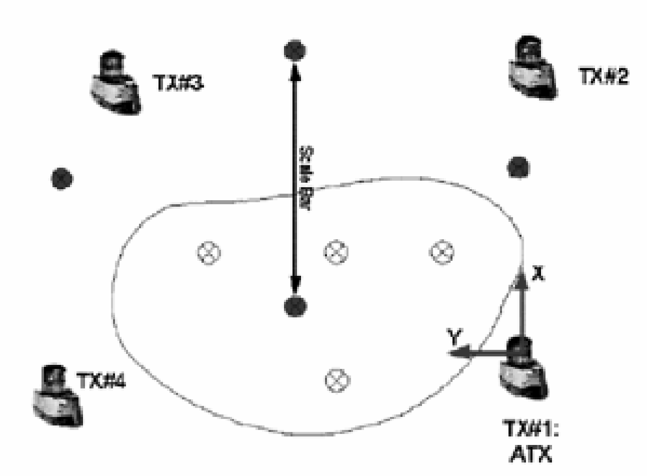


Figure 39 – Typical Arcsecond system setup. The dark dots are suggested calibration points.

Once the system is properly set up, the four transmitters each emit two laser half-planes referred to as laser "fans." These fans are tilted one each at +30^o and -30^o with respect to the vertical. The lasers are mounted in a rotating head and the fans sweep about the vertical axis at the same rate. As shown in Figure 40, as the fans sweep through a point, its angle above the horizontal plane is directly related to the time between the arrivals of the two fans (Arcsecond #063102). The speed of rotation, angular separation of the lasers in the horizontal plane, the tilt angles of the fans, and the arrival times of the fans are necessary in order to determine this vertical angle.

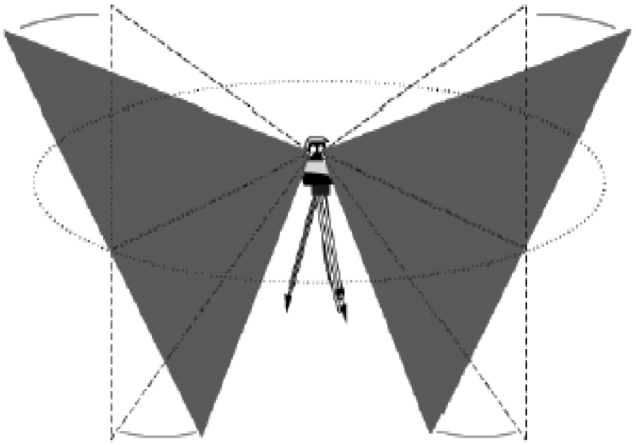


Figure 40 – The two laser fans are tilted in opposite directions.

The horizontal angle can also be determined from the arrival time of either fan given the vertical angle, the speed of rotation, angular separation of the lasers in the horizontal plane, the tilt angles of the fans, and the arrival times of either fan (Arcsecond #063102). In order to synchronize the receiver and transmitter (i.e., define the zero angle), an IR pulse is emitted (see Figure 41).

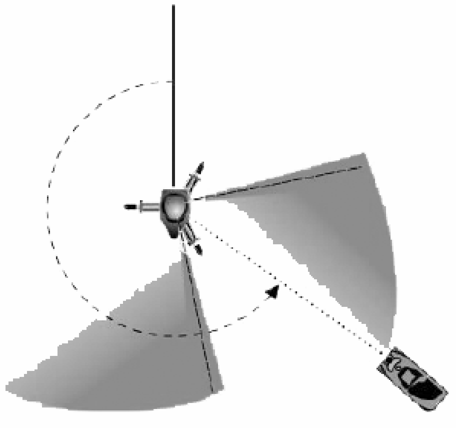


Figure 41 – An IR pulse defines the zero angle reference.

For the coordinates of a receiver to be uniquely determined, three pieces of information are necessary. In this case, the horizontal and vertical angles and the distance from the transmitter are needed. The latter, however, cannot be determined from a single transmitter. Moving the receiver along a line between the receiver and the transmitter would produce the same angular measurements. The angles from at least two transmitters are necessary to uniquely identify a point in 3D space with respect to the transmitters.

D.2 Methodology

There were three main steps in the process to develop a computer model of the Arcsecond system and a preliminary strategy for autonomous calibration. The first was to simulate the action of the transmitters; the second was to derive the position of a receiver given its observed data; and the third was to develop the strategy for calibration. The simulation and analysis were implemented in MatLab script.

D.2.1 Simulation

Imagine a single transmitter at the origin and a single receiver somewhere at [x,y,z], as shown below by the red vector (see Figure 42). One of the laser fans is shown as a green plane. The goal of simulation is to take the known positions of the transmitter and receiver and produce the kind of angular measurements the Arcsecond system would provide.

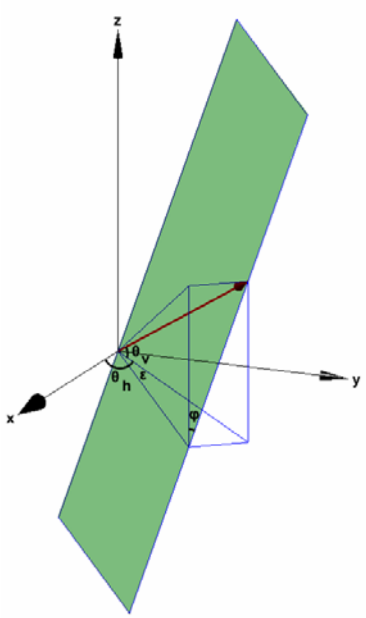


Figure 42 – Geometric model of a laser fan

The vertical angle, θ_v, as a function of the speed of rotation ω, angular separation of the lasers in the horizontal plane θ, the tilt angles of the fans ϕ_{F1}, ϕ_{F2}, and the arrival times of the fans is desired t_{F1}, t_{F2}: $\theta_v(\omega, \theta, \phi_{F1}, \phi_{F2}, t_{F1}, t_{F2})$. Note that for a given transmitter only the times are non-constant. If $\phi_{F1} = \phi_{F2} = 0$, then fans would be vertical and the observed angular delay between the arrival of the fans would be $\theta_{obs} = \theta$. However, the fans are not vertical but tilted approximately 30^{o} in opposite directions. As a result,

$$\theta_{obs} = \theta + \varepsilon_{F1} + \varepsilon_{F2}. \qquad (1)$$

where each fan contributes an offset to the observed delay. Many of the details will be spared, as they do not add to the discussion. Through straightforward trigonometry relations and because the laser fans rotate at a constant rate, it can be shown that

$$\varepsilon_{Fi} = \arcsin(\tan\theta_v \cdot \tan\phi_i), \text{ and} \qquad (2)$$

$$\theta_{obs} = 2\pi\omega(t_{F2} - t_{F1}). \qquad (3)$$

where *i* is the fan index. Combining (1), (2), and (3) yields

$$2\pi\omega(t_{F2} - t_{F1}) = \theta + \arcsin(\tan\theta_v \cdot \tan\phi_1) + \arcsin(\tan\theta_v \cdot \tan\phi_2). \qquad (4)$$

Solving for the vertical angle, gives equation (5):

$$\theta_v(\omega, \theta, \phi_{F1}, \phi_{F2}, t_{F1}, t_{F2}) = \pm\arctan\left(\frac{\sin(2\pi\omega(t_{F2} - t_{F1}) - \theta)}{\sqrt{\tan^2(\phi_1) + 2\tan(\phi_1)\tan(\phi_2)\cos(2\pi\omega(t_{F2} - t_{F1}) - \theta) + \tan^2(\phi_2)}}\right).$$

Given the arrival times of the fans, it is a simple matter to simulate the receiver's vertical angle. Next, a similar derivation must be carried out for the horizontal angle $\theta_h(\theta_v, \omega, \phi_{F1}, \phi_{F2}, t_{F1}, t_{F2})$ given that θ_v is now known. Again, through straightforward trigonometry relations and because the laser fans rotate at a constant rate, it can be shown that

$$\theta_{hFi} = \varepsilon_{Fi} + 2\pi\omega t_{Fi}\text{, and then} \qquad (6)$$

$$\theta_{hFi}(\theta_v, \omega, \phi_{F1}, \phi_{F2}, t_{F1}, t_{F2}) = \arcsin(\tan\theta_v \cdot \tan\phi_i) + 2\pi\omega t_{Fi}\,. \qquad (7)$$

Note that each laser fan generates its own estimate for θ_h. For this simulation, the two estimates were found to be nearly the same and are simply averaged together. Now that the angular data can be calculated, the system can be simulated to find fan arrival times, t_{F1}, t_{F2}.

The MatLab function simFan simulates the action of the laser fans for a particular transmitter as it rotates about the z-axis. The function returns an array of the times when the laser fan intersected the receiver. The simFan operates by rotating the coordinate frame so that the positive x-axis is the fan's intersection with the x-y plane. Hence, a receiver's distance to the fan can be determined by its distance to the positive x-z plane. A receiver's intersection with the fan is found by minimizing the distance from the receiver point to the fan plane. When the distance is zero (rather, when the distance changes signs), a time sample is collected. These time samples are then processed by calcAngles which returns values for θ_h and θ_v based on the fan times. Note that, because the laser planes are tilted at an angle, there is a conical section above and below each transmitter where no reception is possible. These dead zones must be avoided with receivers and calibration points.

D.2.2 Receiver Positions

For a system of four transmitters, the simFan and calcAngles functions are run four times to generate a set of data points. Because the speeds of individual transmitters are not necessarily the same, each transmitter can have a different number of fan times. Each set of fan times generates a corresponding pair [θ_h, θ_v].

If all measurements were perfect, only two transmitters and one set of angular measurements from each would be necessary. However, because the system is subject to a variety of different kinds of noise, a minimum of measurements will not suffice. Instead, multiple readings are taken from four transmitters and the best estimate for the receiver's position is required. Specifically, we want to choose a position for the receiver $\mathbf{p} = [x, y, z]$ such that the expected [θ_h, θ_v] pairs are closest to the observed [θ_h, θ_v] pairs. For the horizontal and vertical angles, the functions

$$\theta_h^{ti} - \arctan\left(\frac{y}{x}\right) \text{ and} \tag{8}$$

$$\theta_v^{ti} - \arctan\left(\frac{z}{\sqrt{x^2 + y^2}}\right) \tag{9}$$

need to be minimized where *t* is the angle from *t*-th transmitter to the receiver and *i* is the *i*-th observed sample from that transmitter-receiver pair. These two equations can be combined into a cost function, *f*, equation (10):

$$f([x, y, z]) = \sum_{t,i} \left[\left(\tan(\theta_h^{ti}) - \frac{y - y_t}{z - z_t} \right)^2 + \left(\tan(\theta_v^{ti}) - \frac{z - z_t}{\sqrt{(x - x_t)^2 + (y - y_t)^2}} \right)^2 \right] \text{ (10).}$$

This cost function is implemented in the MatLab costEvalQuick. The minimization function SolvOpt is then applied, which accepts a guess, and returns the receiver position that minimizes the cost function (more on SolvOpt later). One advantage which this highly over constrained system offers is that initial guesses can be made very easily. This simulation simply uses a minimum set of angles to calculate [x,y,z] for the initial guess. A better routine, which would alleviate outlier issues, would be to average a few minimally calculated [x,y,z]'s together and use that as a guess.

D.2.3 Transmitter Positions

In order to estimate the location of a receiver by minimizing (10), it is necessary to know the positions of the transmitters. In some applications, it might be suitable to simply fix the transmitters and record their positions manually. However, for the purposes of the Trestle research, exactly positioned transmitters are not practical. As a result, we need a system which can estimate even the positions of the transmitters.

In order to accomplish this currently, the Arcsecond system requires that the user collect a set of sample data points from which the transmitter's positions can be determined. Several of these calibration points are required to be at fixed distances from one another. For this project, we would like to be able to collect these data points by automatically programming a robot to run a "calibration" path. Data can then be collected along this path and used to estimate transmitter locations. However, because the robot can only move along the path with some uncertainty, exact measurements will not be possible.

There are several different solutions to this problem. The solution considered here is a natural extension from the estimation of receiver positions. The cost function for transmitter locations given an estimated receiver location is similar to (10). Unlike optimization for the receiver position, however, both the transmitter and receiver positions are considered variables in the cost function. Optimization is not only across the receiver's [x,y,z] coordinate but also across every transmitter's [x,y,z] coordinate. Simply stated, the error is the sum of the errors between each sampled data point and every transmitter.

If r is the r-th sampled point ($1 \le r \le n$), t is the t-th transmitter ($1 \le t \le m$), and i is the i-th angular values between the r-th point and t-th transmitter, then the cost function is, equation (11):

$$f(\mathbf{r}_1, \cdots, \mathbf{r}_n, \mathbf{t}_1, \cdots, \mathbf{t}_m) = \sum_{r,t,i} \left[\left(\tan(\theta_h^{rti}) - \frac{y_r - y_t}{z_r - z_t} \right)^2 + \left(\tan(\theta_v^{rti}) - \frac{z_r - z_t}{\sqrt{(x_r - x_t)^2 + (y_r - y_t)^2}} \right)^2 \right]$$

If there are ten data samples and four transmitters, the cost function must optimize over 42 different variables. While perhaps not the most efficient method, the theory is sound and the SolvOpt function returns acceptable estimates for the transmitter locations.

D.3 Minimization Techniques

The MatLab script uses a non-linear optimization library based on Shor's algorithm. The basis for Shor's algorithm is to take steps in the direction opposite the subgradient at the current point, but in a transformed space. The difference between the current subgradient and the previous subgradient is then used to dilate the space in which the next step is chosen.

Other non-linear optimization processes that show promise for future work are the Levenberg-Marquardt (LM) algorithm and bundle adjustment. The LM algorithm, much like Shor's algorithm, is used to estimate the least squares minimum solution to a non-linear, multi-dimensional equation. Similar results, in terms of accuracy and execution, are estimated for the LM algorithm.

Bundle adjustment is another process that might reduce the dependence of the variables in the transmitter cost function. Bundle adjustment is used in computer vision to produce 3D images from a series of 2D images. Essentially, a minimization process adjusts camera parameters and estimated object positions until the observed and expected feature locations are best matched in the least-squares sense (Bundle Adjustment 2002). Bundle adjustment is named because it adjusts parameters to transform the bundle of light rays projected from features to the camera. Bundle adjustment takes advantage of the fact that the coefficient matrices for the cameras are often sparse and can be reduced to, and solved as, a set of simpler matrices. This kind of optical ray adjustment bears resemblance to the issues faced in Arcsecond triangulation and might be adaptable.

D.4 Summary

When the transmitter positions are exactly known, the receiver positions demonstrated an average error of less than 0.5% (see Figure 43). These errors were the result of the least-squares analysis and errors built into the simulation as described in (Arcsecond #063102).

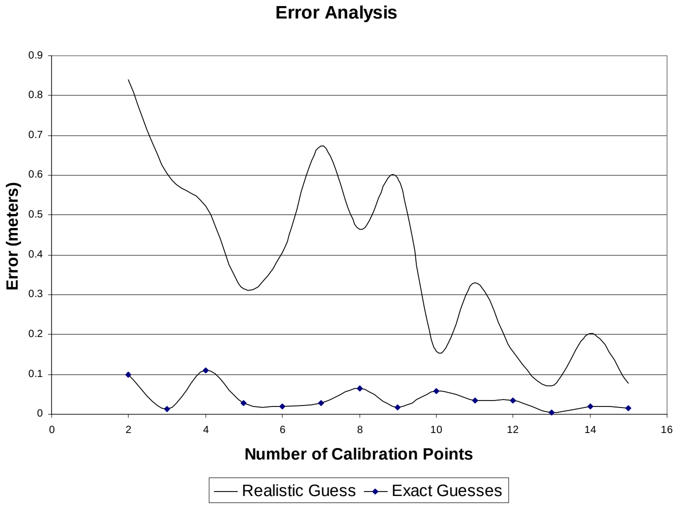


Figure 43 – Plot of error versus number of calibration points.

The new calibration routines demonstrated encouraging results, but more work is needed. For a total of four transmitters and ten sample positions, the average error was about 0.05 meters, with an initial guess that is the exact transmitter location. When the guess is a more realistic estimate of the transmitter positions, the error is around 0.15m. Hopefully, through more advanced minimization techniques, this error can be reduced. The above plot shows the accuracy with and without exact guesses with a variable number of transmitters. For accuracy on the centimeter scale, increasing the number of sample points may not be sufficient.

Appendix E Visual Servoing

The goal of visual servoing is to determine the position and then move towards some location with a robot or robot manipulator using one or more computer-based camera systems. Generally, this target destination will not be known *a priori* and it may be dependent on the location of other objects in the scene. Additionally, the robot's dynamics may be difficult to control or even degrade over time. The final result is that open-loop control of the robot is insufficient and active error correction must be utilized. Visual servoing is typically synonymous with visual feedback/feed-forward control.

In Figure 44, for example, the goal is to *servo* the end-effector, e, to some target location, t. The poses of the cameras *c1* and *c2* relative to the target can be determined using a variety of techniques from computer vision. If the position of the stationary, *c1*, and mobile, *c2*, cameras are known (possibly dependent on position of the end-effector), then the location of the target can be found relative to the world. A similar analysis can be carried out for mobile robots. Rather than determining positions from proprioceptive sensors (e.g., an end-effector pose which is based on joint positions), the robot's pose is usually estimated with exteroceptive sensors and localization techniques.

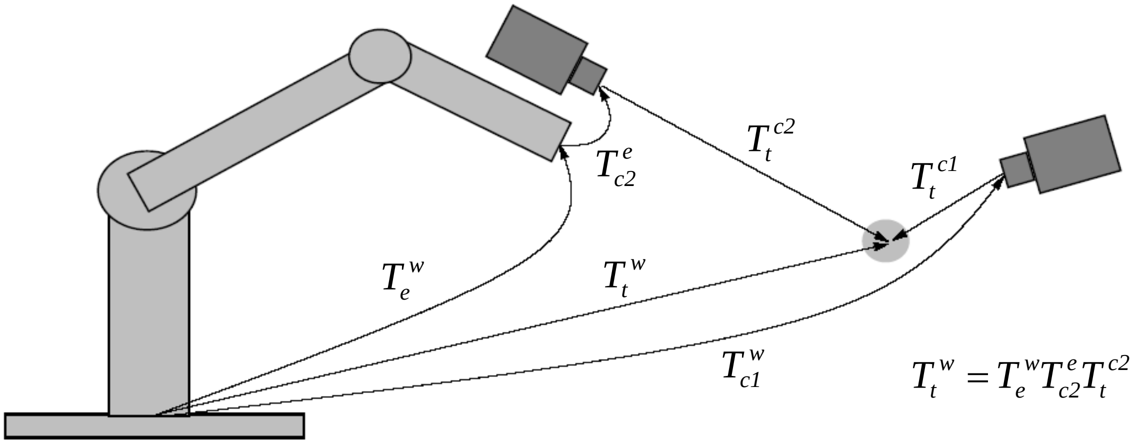


Figure 44 – A typical visual feedback system. If the locations of the target, camera, and end-effector are known, it is possible to calculate a position error term and servo the end-effector toward a target location (Corke, 1996).

First, the following discussion provides a general overview of the major classifications of visual feedback control: image-based and position-based control. These two types of control describe the kind of feedback derived from the vision system. Second, general guidelines for designing control laws are presented. If zero steady-state error is required while tracking a step input, at minimum, a proportional and integral controller is required.

E.1 Architectures

Most vision systems can be classified into two broad categories: image-based and position-based. In an image-based system, the error signal is determined directly from a reference image and the current camera capture. If the reference image contains a bull's-eye in the center, but the bull's-eye is to the right in the image, the end-effector needs to move to the right to correct the difference. Alternatively, in a position-based system, techniques are borrowed from computer vision to estimate some coordinates of the objects in the captured camera image. For example, if the reference coordinates indicate that the bull's-eye needs to be at the origin, but the bull's-eye is 5cm to the right in the current image, the system needs to move 5cm to the right to compensate.

In Figure 45 and Figure 46 there are two feedback loops: one outer loop for the vision system and one inner loop for the robot controller. Most systems include the inner robot controller as both a functional simplification and a means to linearlize the robot's behavior. Several architectures exist that remove this inner controller (Corke, 1996). However, this simplicity makes the feedback loop more unstable.

E.1.1 Image-Based Feedback

As seen in Figure 45, the error in an image-based system is a function of the difference between a reference image and the current image. This difference is typically determined by extracting features from the scene and comparing them with the reference features. The reference image might be generated by manually moving the end-effector to a goal position, capturing an image, and extracting a vector of goal features. In this way, the system is "shown" the desired state.

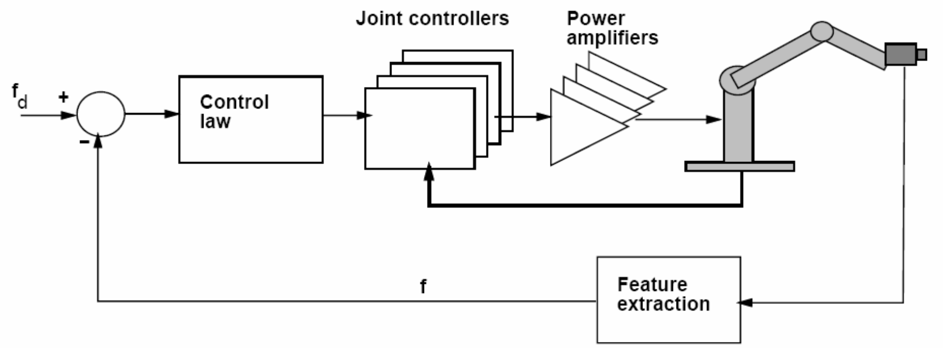


Figure 45 – Image-based visual servoing uses image features to compute an error signal directly.

The movement of the camera can cause complex motion of the image features. For example, as the camera rotates, the image features might move horizontally and vertically. Indeed, one of the consequences of image-based servoing is that the non-linearity makes controller design difficult. This relationship between the actual world and the camera can be modeled with projection equations. The solution is to linearlize the system for very small camera movements. An image Jacobian is defined that relates differential changes in the image features, f, to differential changes in the robot position, r, as shown below. A least squares solution can then be found for the end-effector movements needed to align the reference and current image. Image Jacobians for projective cameras can be found in (Hutchinson, 1996).

$$\dot{f} = \begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix}$$

$$\dot{r} = \begin{bmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

$$\dot{f} = J_v \dot{r}$$

$$\dot{r} = \left(J_v{}^T J_v\right)^{-1} J_v{}^T \dot{f}$$

Equation 1

E.1.2 Position-Based Feedback

The primary advantage to image-based control is that it is relatively simply to implement. However, drawbacks are that it suffers from non-linearities and requires the definition of a reference image which may be unnatural. Position-based feedback takes a different approach, as shown in Figure 46. Extracted features are used to compute the pose of particular objects within the scene. These poses are then compared with reference poses to determine an error signal. With this kind of system, the destination position for the end-effector or robot can naturally be expressed in, for example, Cartesian coordinates.

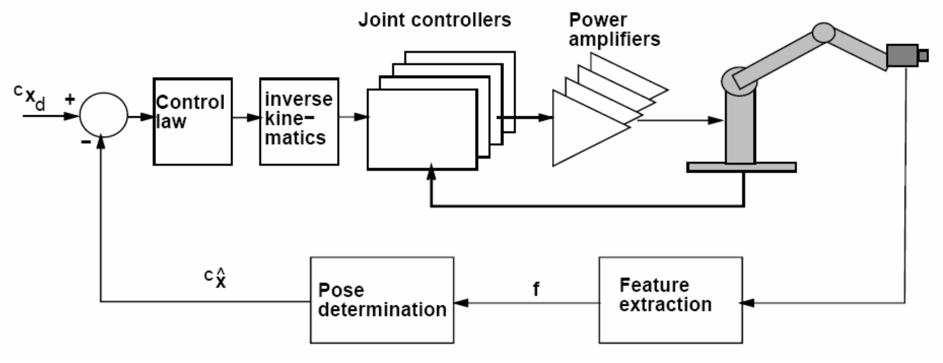


Figure 46 – Position-based visual servoing calculates the pose of objects in the image and uses the results to compute an error signal.

Position-based control is typically more computationally expensive because object poses must be determined relative to the camera. There a variety of techniques to gather pose information from images and it is beyond the scope of this paper to cover them in detail. If, as in the DiRA architecture discussed below, cameras are observing objects with known characteristics, it may be possible to determine position and orientation using a single camera. Stereo vision and depth from motion allow pose estimates to be generated using many camera views (possibly spread over time).

E.2 Implementation Details

A variety of issues related to computer vision also affect visual servoing. Because either the object or the camera (in the case of an end-effector mounted camera) will be moving, the system is likely to experience motion blur. The hardware level solution is to set the camera shutter speed sufficiently fast to prevent any blur. Alternatively, the camera can be *fixated* on the object of interest in the scene. When using fixation, the camera is moved with the object of interest to reduce motion blur.

There are also a variety of camera configurations for visual servoing. Cameras can be mounted directly on the end-effector or robot. However, as the end-effector gets closer to its destination, the camera may become out of focus. The loss of focus may result in decreased accuracy and, essentially, the servoing will become less accurate as it gets closer to its goal where better accuracy is necessary. The camera can also be fixed in the world. Problems may arise if the camera's view is blocked by the robot, but that may be surmountable by using multiple cameras. The camera may also be placed on a second, independent robot. As in the DiRA project discussed later, the result is that the camera can change its focus and move to achieve a better view as the objects change position.

Both types of servoing require the extraction of feature images. A thorough discussion is beyond the scope of this paper, but classification by color, optical flow, and edge and corner detection are all candidate techniques.

E.3 Control Laws

A control law determines what commands should be sent to the robot given some error between the goal and current positions. A visual servoing system is a discrete time system and, like any other discrete time system, can be analyzed using frequency domain techniques. It is then possible to evaluate the system's performance as a function of the movement of the target. A situation might arise where the robot needs to track a static target, a linearly moving target, or a target moving along a parabolic path.

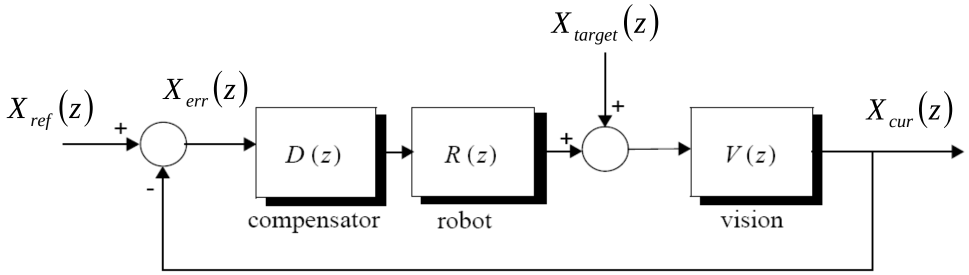


Figure 47 – A system model for a visual servoing system. Movement of the target can be considered a disturbance to the system.

In Figure 47, a reorganized view of the system is shown to illustrate a disturbance analysis. In this case, the movement of the target, $X_{target}(z)$, acts as a disturbance to the system. A static target location corresponds to a step input, a linearly moving target to a first order input, and a parabolic path to a second order input at $X_{target}(z)$.

$$X_{target}(z) = \begin{cases} \dfrac{z}{z-1} & \text{step target} \\ \dfrac{Tz}{(z-1)^2} & \text{ramp target} \\ \dfrac{T^2 z(z+1)}{(z-1)^3} & \text{parabolic target} \end{cases}$$

Equation 2

Without loss of generality, assume that the reference input is constant zero. The transfer equation between target motion and error can be written as:

$$\frac{X_{err}(z)}{X_{target}(z)} = \frac{V(z)}{1+V(z)R(z)D(z)}$$

Equation 3

Then, using the final value theorem, the system's steady state error, $\underset{t\to\infty}{Lim}\, x_{err}(t)$, can be evaluated. For a given target movement, it is generally desired that the steady state error go to zero (i.e., the end-effector tracks the target perfectly given enough time):

$$\begin{aligned} \underset{t\to\infty}{Lim}\, x_{err}(t) &= \underset{z\to1}{Lim}(z-1)X_{err}(z) \\ &= \underset{z\to1}{Lim}\frac{(z-1)V(z)}{1+V(z)R(z)D(z)}X_{target}(z) \end{aligned}$$

Equation 4

For the steady state error to be zero when a particular $X_{target}(z)$ is substituted from Equation 2, there must be at least one zero at *z=1*. (Note that the single zero will always cancel out for the example target motions.) In general, the vision, *V(z)*, and robot, *R(z)*, system transfer functions will not be known. It is possible that with only a proportional controller the system will reach zero steady state error. However, to guarantee zero steady state error, one or more poles can be engineered into the controller, *D(z)*. Recalling that a pole at *z=1* is an integrator; most systems will require at least an integrator in the controller. Additional derivative and proportional controller terms will further increase system response.

References

Arcsecond. Error Budget and Specifications. White Paper #063102.

Arcsecond. Indoor GPS Technology for Metrology. White Paper #071502.

Barber, K.; Martin, C.; Reed, N.; and Kortenkamp, D. Dimensions of Adjustable Autonomy. In Advances in Artificial Intelligence: PRICAI 2000 Workshop Reader (R. Kowalczyk, S. W. Loke, N. Reed, and G. Williams, eds.), vol. 2112, pp. 353--361, Berlin: Springer Verlag, 2001.

Berry, P.; Gervasio, M.; Uribe, T.; Myers, K; and Nitz, K. A Personalized Calendar Assistant. In proceedings of the 2004 AAAI Spring Symposium on Interaction between Humans and Autonomous Systems over Extended Operation.

Blum, A. and Furst, M.. Fast planning through planning graph analysis. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp 1636-1642. Montreal, Canada. Morgan Kaufmann, 1995.

Brainov, S. and Hexmoor, H. Quantifying Relative Autonomy in Multiagent Interaction. The IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, 2001.

Bruce, J. and Veloso, M. Fast and accurate vision-based pattern detection and identification. Proceedings of ICRA-03, the 2003 IEEE International Conference on Robotics and Automation (ICRA '03), May, 2003.

Bundle Adjustment. http://www.esat.kuleuven.ac.be/~pollefey/tutorial/node76.html. November 5, 2002.

Cass, S. “Can the Hubble Telescope be Saved?” IEEE Spectrum. March 2004.

Chen, I. Stochastic Petri Net Analysis of Deadlock Detection Algorithms in Transaction Database Systems with Dynamic Locking. The Computer Journal, 1995.

Cimatti, A.; Rovero, M.; and Traverso, P. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In Proceedings of AAAI-98.

Corke, P. Visual Control of Robots: high-performance visual servoing. John Wiley & Sons, Inc. New York, 1996.

Corkill, D.; Rubinstein, Z.; and Lander, S. Coordination of Human and Software Agents in Time and Resource-Constrained Dynamic Processes. Workshop on Humans and Multi-Agent Systems. AAMAS-03 Melbourne, Australia. July, 2003

Dasgupta, B. and Mruthyunjaya, T. A Canonical Formulation of the Direct Position Kinematics for a General 6-6 Stewart Platform. Mechanical Machine Theory. Vol 29, 1994.

David, R. and Alla, H. Petri Nets and Grafcet. New York, Prentice Hall, 1992.

Dennis, J.E., Jr. “Nonlinear Least Squares,” State of the Art in Numerical Analysis, ed. D. Jacobs, Academic Press, pp. 269-312, 1977.

Doggett, W. A Guidance Scheme for Automated Tetrahedral Truss Strucutre Assembly Base don Machine Vision. NASA Technical Paper 3601, November 1996.

Dorais, G.; Bonasso, R.; Kortenkamp, D.; Pell, P.; and Schreckenghost, D. 1998. Adjustable autonomy for human-centered autonomous systems on mars. Presented at the Mars Society Conference.

Dutre, S.; Bruyninckx, H.; and Schutter, J. The analytical Jacobian and its derivative for a parallel manipulator. IEEE International Conference on Robotics and Automation. 1997.

Fleming, M. and Cohen, R. A utility-based theory of initiative in mixed-initiative systems. The IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, 2001.

Fong, T.; Cabrol, N.; Thorpe, C.; and Baur, C. A Personal User Interface for Collaborative Human-Robot Exploration. In Proc. International Symposium on Artificial Intelligence, Robotics, and Automation in Space. Montreal, Canada. June, 2001.

Fong, T.; Thorpe, C.; and Baur, C. Robot, Asker of Questions. In Robotics and Autonomous Systems. Vol. 42, 2003.

Fox, D.; Burgard, W.; Dellaert, F.; and Thrun, S. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. AAAI/IAAI 1999.

Gillespie, R.; Colgate, J.; and Peshkin, M. A General Framework for Cobot Control. International Conference on Robotics and Automation. Detroit, MI. 1999.

Goodrich, M.; Crandall, J.W.; and Stimpson, J. L. Neglect Tolerant Teaming: Issues and Dilemmas. In proceedings of the 2003 AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments.

Griffis, M. and Duffy, J. A Forward Displacement Analysis of a Class of Stewart Platforms. Journal of Robotic Systems. Volume 6, 1989.

Hager, Hutchinson and Corke. A tutorial (TT3) on visual servo control run at the IEEE Robotics & Automation Conference 1996.

Han, F. and Veloso, M. Physical Model Based Multi-objects Tracking and Prediction in Robosoccer. In Working notes of the AAAI 1997 Fall Symposium on Model-directed Autonomous Systems, MIT, Boston, November 1997.

Han, K. and Veloso, M. Reactive Visual Control of Multiple Non-Holonomic Robotic Agents. In Proceedings of the International Conference on Robotics and Automation, Belgium, May 1998.

Herstrom, C.; Grantham, C.; Allen, C.; Doggett, W.; et al. Software Design for Automated Assembly of Truss Structures. NASA Technical Paper 3198, June 1992.

Hoffman, J. and Nebel, B. The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research. Vol 14, 2001.

Huang, T. and Yap, T. Singularity of Parallel Manipulators. www.ntu.edu.sg/mpe/Research/Projects/YapKianTiong/Sing_for_J.html.

Jakobovic, D. and Jelenkovic, L. The Forward and Inverse Kinematics Problems for Stewart Parallel Mechanisms. University of Zagreb.

Jensen, K. Coloured Petri Nets. Springer, Germany, 1996.

Jensen, K. Coloured Petri nets: basic concepts, analysis methods, and practical use. New York, Springer, 1996.

Jensen, K. and Rozenerg, G. High-level Petri Nets. Springer-Verlag, Germany, 1991.

Jensen, R. and Veloso, M. OBBD-based Universal planning for synchronized agents in non-deterministic domains. Journal of Artificial Intelligence Research, 13, pages 189-226, 2000.

Kim, H. and Choi, Y. The Kinematic Error Bound Analysis of the Stewart Platform. Journal of Robotic Systems. Volume 17(1), 2000.

Khatib, O. Force Strategies for Cooperative Tasks in Multiple Mobile Manipulation Systems, In Proc. International Symposium of Robotics Research, Munich. October, 1995.

Kortenkamp, D.; Bonasso, R.; Ryan, D.; and Schreckenghost, D. Traded Control with Autonomous Robots as Mixed Initiative Interaction AAAI Spring Symposium on Mixed Initiative Interaction , March, 1997.

Kortenkamp, D.; Burridge, R.; Bonasso, P.; Schrenkenghoist, D.; and Hudson, M. An intelligent software architecture for semi-autonomous robot control. In Autonomy Control Software Workshop, Autonomous Agents 99, 1999.

Kortenkamp, D. Designing an Architecture for Adjustably Autonomous Robot Teams. In PRICAI Workshop Reader, LNAI 2112 , eds. R. Kowalcyk, S. W. Lake, N. Reed, and G. Williams, Springer-Verlag, New York, 2001.

Kortenkamp, D.; Schreckenghost, D.; and Martin, C. User Interaction with Multi-Robot Systems, in Workshop on Multi-Robot Systems, 2002

Lenser, S. and Veloso, M. Sensor Resetting Localization for Poorly Modeled Mobile Robots. Proceedings of the IEEE International Conference on Robotics and Automation, 2000, 2000.

Lenser, S. and Veloso, M. Visual Sonar: Fast Obstacle Avoidance Using Monocular Vision. In Proceedings of IROS’03, 2003.

Martin, C.; Schreckenghost, D.; and Bonasso, P. Augmenting Automated Control Software to Interact with Multiple Humans. In proceedings of the 2004 AAAI Spring Symposium on Interaction between Humans and Autonomous Systems over Extended Operation.

Martin, C.; Schreckenghost, D.; Bonasso, P.; Kortenkamp, D.; Milam, T.; and Thronesbery, C. Distributed Collaboration Among Humans and Software Control

Agents. Proceeding of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS 2003, NARA, Japan, May 19-23, 2003.

Marquardt, W. “An Algorithm for Least-Squares Estimation of Nonlinear Parameters.” Journal of the Society for Industrial and Applied Mathematics, Vol. 11, Issue 2 (Jun 1963), 431-441.

Merlet, J. Parallel Robots. Kluwer Academic Publishers, London. Volume 74, 2000.

Miller, C.; Funk, H.; Dorneich, M.; and Whitlow, S. A Playbook Interface for Mixed Initiative Control of Multiple Unmanned Vehicle Teams.

More, J.J. “The Levenberg-Marquardt Algorithm: Implementation and Theory.” Numerical Analysis, ed. G.A. Watson, Lecture Notes in Mathematics 630, Springer Verlag, 105-116.

Rhodes, M.; Will, R.; and Quach, C. Baseline Tests of Autonomous Telerobotic System for Assembly of Space Truss Structures. NASA Technical Paper 3448, July 1994.

Scerri, P. and Pynadath, D. Towards Adjustable Autonomy for the Real World (2002).

Schreckenghost, D.; Bonasso, P.; Martin, C.; Milam, T.; and Thronesbery, C. Human-Agent Teams for Extended Control Operations in Space. In proceedings of the 2004 AAAI Spring Symposium on Interaction between Humans and Autonomous Systems over Extended Operation.

Shin, D. H.; Hamner, B.; and Singh, S. Motion Planning for a Mobile Manipulator with Imprecise Locomotion.

Sierhuis, M.; Bradshaw, J.; Acquisti1, A.; Hoof, R.; Jeffers, R.; and Uszok, A. Human-Agent Teamwork and Adjustable Autonomy in Practice. Proceeding of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS 2003, NARA, Japan, May 19-23, 2003.

Simmons, R. and Apfelbaum, D. A Task Description Language for Robot Control. Proceedings Conference on Intelligent Robotics and Systems, October, 1998

Simmons, R.; Singh, S.; Hershberger, D.; Ramos, J.; and Smith, T. Coordination of heterogeneous robots for large-scale assembly. In Proceedings of the 7th International Symposium on Experimental Robotics (ISER). 2000.

Simmons, R.; Smith, T.; Dias, M.; Goldberg, D.; Hershberger, D.; Stentz, A.; and Zlot, R. A Layered Architecture for Coordination of Mobile Robots. In Multi-Robot Systems: From Swarms to Intelligent Automata, A. Schultz and L. Parker (eds.). Published by Kluwer, 2002.

SolvOpt, MatLab Script. http://www.kfunigraz.ac.at/imawww/kuntsevich/solvopt/. November 5, 2002.

Steele, J.; Debrunner, C.; Vincent, T.; and Whitehorn, M. Developing stereovision and 3D modeling for LHD automation. 6th International Symposium on Mine

Mechanization and Automation, South African Institute of Mining and Metallurgy, 2001.

Stein, M. and Paul, R. Behavior Based Control in Time Delayed Teleoperation. In Sixth International Conference on Advanced Robotics, 1993.

Veloso, M.; Stone, P.; Han, K.; and Achim, S. The CMUnited-97 Small-Robot Team. In Hiroaki Kitano, editors, RoboCup-97: Robot Soccer World Cup I, pp. 242–256, Springer Verlag, Berlin, 1998.

Veloso, M.; Stone, P; Han, K.; and Achim, S. CMUnited: A Team of Robotic Soccer Agents Collaborating in an Adversarial Environment. In Hiroaki Kitano, editors, RoboCup-97: The First Robot World Cup Soccer Games and Conferences, pp. 242–256, Springer Verlag, Berlin, 1998.

Veloso, M.; Uther, W.; Fujita, M.; Asada, M.; and Kitano, H. Playing Soccer with Legged Robots. In Proceedings of IROS-98, Intelligent Robots and Systems Conference, Victoria, Canada, October 1998.

Wannasuphoprasit, W.; Akella, P.; Peshkin, M.; and Colgate, J.E. Cobots: A Novel Material Handling Technology. International Mechanical Engineering Congress and Exposition. Anaheim, ASME 98-WA/MH-2, 1998.

Ward, R.; Law, J.; and Bloomquist, L. Remote Control System. United States Patent #6,633,800. October 14, 2003.