From Teleoperation to Autonomy: “Autonomizing” Non-Autonomous Robots

B. Kievit-Kylar[1], **P. Schermerhorn**[1], **and M. Scheutz**[3]
[1]Cognitive Science Program, Indiana University, Bloomington, IN 47404, USA
[2]Department of Computer Science, Tufts University, Medford, MA 02155, USA

Abstract—*Many complex tasks (search and rescue, explosive ordinance disposal, and more) that eventually will be performed by autonomous robots are still performed by human operators via teleoperation. Since the requirements of teleoperation are different from those of autonomous operation, design decisions of teleoperation platforms can make it difficult to convert such platforms for autonomous use. In this paper, we discuss the differences and the potential difficulties involved in this conversion, as well as strategies for overcoming them. And we demonstrate these strategies by “autonomizing” a fully teleoperated robot designed for tasks such as bomb disposal, using an autonomous architecture that has the requisite capabilities.*

Keywords: tele-operation, autonomy, conversion

1. Introduction

Teleoperated robots are becoming widely available for all kinds of activities, from defusing bombs (like the IRobot Packbot in war zones)[1] to remote presence robots (like the Vgo)[2]. These robots are able to go places and do things that would normally be too dangerous for humans, or simply be too expensive to be practical. They include a range of sophisticated equipment and low-level control algorithms, and would seem to be naturally suited, and are certainly highly desirable, for autonomous operation. Compared to autonomous robots, teleoperated robots are easier to build and easier to sell because no control system for autonomous operation has to be designed (all the intelligence and control lies with the human operator instead) and sometimes poorer quality or fewer sensors can be used (as humans can generally make better use of available sensors than current autonomous techniques). Hence, it is unsurprising that increasing the efficacy of teleoperation interfaces is a continued research focus (e.g., [1], [2], [3]).

One obvious drawback of teleoperation is the need to have a live control stream of data for sensors and effectors. However, given the constraints that make robots appealing tools, it may be difficult or impossible to establish the data connection (e.g., because of the distances involved or environmental interference), and requiring a controller to be close to the robotic platform may put the operator in danger, eliminating the primary advantage of using a robot. Even in the absence of such obstacles, the need for constant human attention adds expense and limits the overall effectiveness of the robot. The ability to operate these robots autonomously would avoid many of these limitations, but researchers need access to the robots to continue to improve the performance of autonomous robot architectures to the point where the human operator is not needed.

The common response to the above issues has been the design of “hybrid autonomous/teleoperated” systems that combine the strengths of both approaches. Mixed-initiative architectures are suitable as extraplanetary rover control systems, where prohibitive communication delays necessitate some degree of autonomy, while the higher-level decision-making remains in the hands of a team of human experts [4]. Several groups have examined effective mechanisms for adjustable autonomy, so that the balance between teleoperation and autonomy can be dynamically altered to fit the circumstances [5], [6], [7].

While mixed-initiative systems touch on the autonomization problem, these systems are designed from the ground up with partial autonomy in mind. Very little work has been done on converting systems that were designed to be purely teleoperated into partially or fully autonomous systems. Not only is the literature on the transformation of non-autonomous robots into autonomous ones far sparser, but the existing papers also focus mostly on the integration of one particular robotic platform. E.g., Barreto [8] describes the autonomization of the ANDROS platform using the idea of interchangeable “brick” components to add new sensory capabilities. Similarly, Stewart and Khosla[9] describe a modular control mechanism for a Puma 560 in which a teleoperation module is replaced with an autonomous one.

We believe that a greater understanding of how a conversion can be done would be greatly beneficial to the field. In this paper, we describe our efforts for developing systematic principles for converting non-autonomous teleoperated robots into fully autonomous platforms. We start with a brief background on teleoperated and autonomous robots. We then discuss some of the design philosophies that affect “autonomizing” teleoperated systems. Next we describe our transformation of the Multi-Mission Payload Platform (M2P2) into a general

[1]`http://www.irobot.com/gi/ground/510_PackBot`
[2]`http://www.vgocom.com/what-vgo`

purpose autonomous system. After presenting an evaluation of the new system in a "bomb disposal" task, we finish with some discussion about limitations of this technique as well as how generalizable it is to other robotic platforms (`http://www.apitech.com/products/multi-mission-payload-platform-m2p2`).

2. "Autonomizing" Teleoperated Robots

Successfully implementing an autonomous architecture on a robot that has been designed for teleoperation requires overcoming a variety of challenges. All of these challenges are traceable in some way to how different constraints factor into the design of purely teleoperated robots.

While there are some obvious mechanical differences existing in the hardware needed to perform communication with an operator verses an autonomous system, many more subtle assumptions go into the construction of these two types of robotic platforms. Autonomous and teleoperated robots will often require different levels of sophistication in the sensors needed to perform their tasks. For example, while the output of wheel encoders and range-finders may be very useful in allowing a program to determine its position, often humans rely on more expensive sensors such as video cameras to quickly expose them to a high volume of noisy data. As well as sophistication in sensors, complex robots with many moving parts require complex control systems. While a human operator may only be able to focus on a subset of controls at any given time, microchips are able to compartmentalize the various effectors allowing simultaneous meaningful control of all. While more user friendly interfaces or multiple operators may alleviate the control problem to some degree, it can not match the orchestration of a single programmatic control architecture. Similarly, the computational requirements of a remotely-controlled robot can be very low. Hence, there may be no general-purpose onboard computer, or that computer may be underpowered to handle the tasks (problem-solving, sensor processing, etc.) required for autonomous operation.

The same goes for effectors; for example, the design of a gripper that will be actuated via a gamepad in the hands of a skilled user might look very different from what an autonomous robot architect might ideally envision (e.g., because it has fewer safeguards, relying instead on the operator's skill and judgment). Even at the mundane level of connectivity, different requirements are evident; a relatively low-bandwidth wireless connection might be acceptable for teleoperation (e.g., lower frame-rates, lower resolution, or higher noise in a camera feed may be easier for a human operator to deal with than for a vision processing algorithm; similarly for audio streams and voice recognition). Autonomous operation may, therefore, necessitate the addition or replacement of computational, sensing, or effecting equipment on the teleoperation platform.

We now address specific challenges:

a) Physical connection: Sensors and effectors need to be exposed to the control architecture. In some cases it may be possible to make use of the existing teleoperation channel, but (as noted above), this will sometimes not be feasible. In that case, it will be necessary to create alternative interfaces (e.g., RS-232 or USB-based serial or wired Ethernet connections). Moreover, if device drivers and libraries do not provide access at the right level of abstraction, it may be necessary for the control architecture to implement a software interface, as well.

b) General control algorithms for effectors: Similarly, depending on the nature of the particular sensors and effectors, it might be possible to use existing control algorithms (e.g., for a mobile robot base or for an n-degree of freedom manipulator) or to use them after only minor adaptations. In the worst case (typically because a given piece of hardware is used differently under teleoperation than under autonomy), new algorithms need to be developed for particular sensors and effectors. For example, feedback from sensors such as wheel encoders can be used to generate a simple, immediate, computational understanding of an autonomous robot's pose. However, the human operator is unlikely to find encoder counts useful, so they may not be exposed via a software interface, and in the worst case, may not even be present at all. Other sensors, such as cameras, are often easier to access, but may still require additional processing steps (e.g., to extract individual frames for visual processing from a video stream encoded for efficient transmission).

c) Control architecture: Naturally, one of the biggest changes will be the inclusion of the autonomous control architecture itself. A wide variety of configurations is possible; depending on the task, the control architecture will comprise different components, possibly including a planner, an action execution component, components for user interfaces including screen-based and possibly spoken natural language dialog interfaces. The task approach can either be pre-programmed (e.g., in terms of action scripts) or the robot can be dynamically instructed during task execution. The latter requires the control architecture to use problem-solving for new tasks for which no action scripts exist [10].

d) "Plug-and-play" generalizability: Adding an existing control system can be relatively straightforward if it implements and uses a well-defined API structure. Typically, sensors will fall under a particular sensing category, e.g., range sensors, vision sensors, force sensor, etc., thus allowing for a generic interface to be adapted for the particular sensor use. Similarly, typical effectors such as wheels for mobile platforms, grippers, or robotic arms will allow for abstractions that map effector-specific commands and properties to more generic interfaces that can interact with standard algorithms. If the control system is designed with such generic interfaces in mind, the problem can be viewed as one of ensuring that the control system's expected interfaces are mapped properly onto the teleoperated robot's hardware and software interfaces. The difficulty here is that there is unlikely to be a one-to-one match of functional units between the teleoperation platform and the control architecture. For example, the robot's interface may support driving arm joint motors at a given velocity, in agreement with the control architecture's expectations. However, it may lack a predefined position-based joint motion mechanism, which must be added as part of the integration if the control architecture expects it. Similarly, there is no reason to assume that both architectures will use the same relative sign or units.

e) Programatic interface: Ideally, whether it is implemented as a software library (e.g., C/C++ shared objects, Java classes, etc.), simply given as a communication protocol (e.g., custom serial protocols, standardized infrastructures, etc.), or some combination of the two, the interface for a new piece of hardware will be well-specified and clearly documented. However, given that teleoperated robot platforms are not originally intended to be targets for development, the likelihood of undocumented features or errors in the specification is somewhat higher than normal. In such cases, it may be necessary to examine how the teleoperation controller interfaces with the robot architecture that it was built with. Studying the effects of (or responses to) valid messages sent from the original teleoperated controls, can be very helpful for "filling in" gaps. These messages can then either be analyzed for patterns or stored as a response lookup table to be used later by the software.

f) Simulation: Simulation is a valuable tool for any autonomous robot architecture, and not just to facilitate testing before deploying on the physical robot. Take, for example, the case of arm movement and manipulation. Since human operators inherently have extremely rich mental models of the systems that they are working with, they do not require precise measurements of all of the possible positions in joint space. Simply by viewing a handful of video frames taken from a mounted camera, the operator can determine approximately how that camera is mounted in relation to the robot and the target as well as determining an appropriate movement path. For the autonomous architecture, on the other hand, pre-building a highly accurate model of the known environment (the robot and its manipulators) will allow it to avoid the time-consuming process of building relational models on the fly. Such simulation planning also allows a robotic architecture to plan movements through space that optimally avoid obstacles, which is especially important in cases where feedback from the platform regarding joint positions is of less than ideal fidelity.

2.1 Experimental Platform

The robot we chose to autonomize is the APITech Multi-Mission Payload Platform (M2P2). This robot was designed to be controlled by a human operator primarily via a wireless XBox game controller. The M2P2 is a three wheeled holonomic robot with each wheel capable of rotating independently. It is equipped with a 2 DOF arm terminated by a 2 DOF gripper. The operator can get the robot's view of its environment through two cameras, one mounted on the platform and one mounted on the arm. In addition, the platform includes both a speaker and a microphone, allowing the operator to speak through the robot and hear any sounds near the robot.

Communication in the M2P2 is a subset of JAUS (the *Joint Architecture for Unmanned Systems*, a standard for open interoperability in robotic architectures [11]), augmented by a small number of custom message types. The M2P2 implements a set of core movement functionality as well as including custom messages for the robot arm control and specific sensors. JAUS messages on this robot are passed on two onboard networks, one for sensors and one for effectors. Messages are transmitted wirelessly to a pack based Operator Control Unit (OCU). Video streams are played back on a dedicated tablet PC along with custom built software to facilitate control of robot movement.

We integrated the M2P2 into ADE (the *agent development environment*), a Java-based infrastructure for developing robotic architectures [12], [13], [14]. ADE provides communication mechanisms that allow modular functional components to interact in a distributed fashion using Java remote method invocations. ADE components can register their own services and look up services provided by other components, and connections between components are monitored to ensure robust execution. A well-defined suite of programming interfaces allows consistent access to a variety of sensors (e.g., laser range-finders, GPS sensors, and cameras) and effectors (e.g., robot bases such as the Segway RMP, Mobilerobots Pioneer, and Videre Era). Implementing these interfaces on the M2P2 allows us to take advantage of the existing autonomy facilities in ADE, including problem-solving and goal management.

2.2 Communication

The first step in autonomization is to establish communication pathways between the teleoperated robot base and the computer that will host the autonomous architecture. This meant bypassing the existing wireless RF interface used by the teleoperation rig to instead make direct (wired) Ethernet connections to the robot.

Once the physical connection has been made, it must be tested. Before the cheap availability of high performance micro processors and network adapters, connection types were likely all proprietary and highly compact. Now, developers are moving toward higher bandwidth communication to help improve comprehension of the messaging system. Protocols like the Internet Protocol (IP) both internally provide a lot of functionality (message ordering, routing, etc.) but also have many tools to help receive, send and process information. Robots running such protocols can often be tested with packaged communication processing tools, while rarer protocols may require testing specific to a given communication design.

g) Communicating with Effectors: Although communications protocols could theoretically be arbitrarily complex, they break down into two parts, syntax and semantics. For each functional ability to be run on the robot, syntax is the set of argument variables with possibly some information on their size and type, and semantics is the generalized understanding of how the bits in each argument are interpreted (as flags, as integers representing feet, doubles representing miles per hour, etc.). When trying to back-engineer a communication protocol, the syntax of the command must be determined first. The syntax alone however is not sufficient for interpreting and generating commands of that type as the variables must be unpacked as values and then those values given in meaningful units or conditions.

For example, when implementing robot arm control for the M2P2, we determined from the JAUS protocol that the command to modify the arm position was a concatenation of values each representing a velocity of one of the arm joints. This information alone was insufficient even to determine the exact syntax of the command. Through feedback from the manufacturer in conjunction with systematic testing of options, and validation of the specifications, we found the ordering of the joints and number of bytes allocated to each joint in the command. While this specified the syntax of the commands, it was still unclear how each set of around four bytes (more for some joints and less for others) translated into a velocity on the physical arm.

Some of the most common encoding variants are; flags, scaled integers, IEEE floating point or doubles, and angles. Flags can be tested through simple trial and error when the outcome of the flag is an obvious physical change in the system. Scaled integers (where the values are interpreted as integer values times a scaler plus an offset), are often easy to detect as similar integer input values will lead to similar output speeds or other output types. Floating point or double precision number should be tested similarly to scaled integers, where similar inputs cause similar outputs however instead of using similar integer representations to test, convert the values into either doubles or floating points. Angles, while a subset to scaled integers or decimal precision numbers, are a special case as they are cyclic in nature and this must be accounted for when trying to determine if a particular encoding is of this form.

When all attempts at understanding the semantics fail (i.e., a systematic test of the possible input space provides no observable consistences), an alternative approach is to build a simple input/output lookup table. Recording what direction a particular joint makes and potentially some gross speed conditions, for a set of given inputs, can allow a simply designed inverse process where a desired speed can be achieved by finding the closest speed known in the lookup table. Some times, it is also sufficient to have the robot operate at a single speed (or more often, one speed in both the positive and negative direction) in which case a hard coded single value can be used.

The JAUS protocol specifies two main types of movement commands: "set velocity" and "move relative". Because this platform was designed to be controlled by a remote human operator, only the "set velocity" commands are implemented. Similarly, although the wheel units almost certainly include wheel encoders, they are not accessible via the JAUS interface. This was a common theme to the design, where simplifying steps were used in terms of what to expose to the external system based on the assumption that the robot would not operate autonomously. However, with the "set velocity" commands, we were able to implement all of the elements of the movement interface expected by the control architecture.

Teleoperated robots are often designed to expect continuous command feedback from an operator. The robot may receive a constant stream of drive commands even though the operator has held the movement joystick at a constant angle. Since the robot can not correct its error if the human operator is disconnected, if no new commands are received in a given amount of time, most designs will stop the robot movement. While this may be logical for an autonomous system as well, there is a stronger bias in a teleoperated system to default to no action and let the human override if this is a problem. While converting a teleoperated system this is important to note as it may require additions such as message repetition at constant intervals to keep the robot performing a desired action.

h) Communicating with Sensors: One of the most important sources of information for an autonomous robot is the camera; we will illustrate the process of establishing communication with a sensor using the example of interfacing with the M2P2 camera. From the programmer's perspective, it would be easiest if it were possible to simply grab fully-formed individual frames from the video stream. However, such an encoding scheme would consume too much bandwidth in many domains, so a variety of protocols take advantage of similarities between adjacent frames to compress the data stream. Moreover, although many of these video encoding protocols are well-known, in some cases proprietary variations may have been added to the standard protocol to meet the needs of the particular robotic platform. This imposes additional challenges when converting a teleoperated robot for autonomous operation, as even the base protocol might be unknown, and any proprietary extensions are unlikely to be published.

As noted above, the main challenge for utilizing the cameras is deciphering the video stream. Primary variations in video streams to be aware of at this stage are, different color models (RGB, CMYK, etc) and streaming verses static video translation (some decoders require the full video buffer to be available before decoding can begin, this will not work

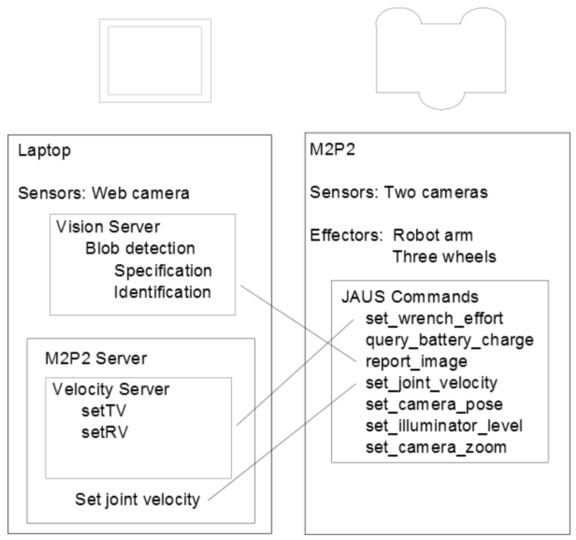

Fig. 1: **Architecture layout of the M2P2 and controller laptop.**

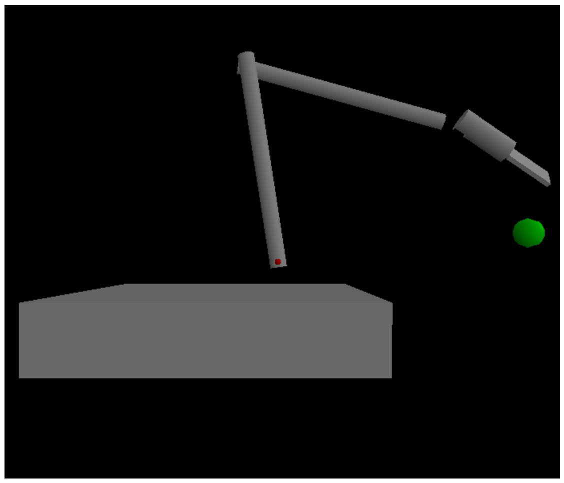

Fig. 2: **Visualization of the robot arm simulator.**

for streaming video feed from a robot).

When decoding the M2P2 video format, we started by searching the data for likely protocol elements, such as sequential markers added to verify order of packets received. This allowed us to discover the higher fidelity "key frames" that were sent at regular intervals to allow cumulative error between frames to be reset. In this particular case, we were able identify the protocol using the key frames and apply standard decoding routines to extract frames from the stream. We determined by the data header, that the M2P2 used a slightly augmented mpeg encoding variant. After removal of the additional formatting, the video stream could be parsed into readable raster images, using the FFMPEG decoder.

2.3 System Configuration and Control

Once basic communication protocols had been established, the system was integrated with the ADE infrastructure. A general purpose JAUS component was written to facilitate creation and interpretation of JAUS messages. Figure 1 shows the connections between the M2P2 control architecture and the architecture on the laptop used to control the newly autonomous robot.

The newly added laptop served as the primary intelligence control for the robot platform. As the M2P2 server is fully ADE compliant, it has access to all other ADE servers and functionality available to other holonomic robots in that environment. This includes a vision server to process incoming camera feeds, planners, and natural language interpreters. The M2P2 becomes a removable module fitting into a larger autonomous robotic infrastructure. As well as adding functionality, the ADE infrastructure adds error handling, security and the ability to offload processes not only from the original robots processors but also from those physically added to the robot. A smaller, less capable control unit can then be used to interface with the robot while the computationally expensive operations such as image processing can be handles by external, more powerful computers.

A web camera was connected to the laptop directly and mounted on the front of the mobile platform. While the robot had built in video feeds, these were slow, of lower quality, and computationally consuming to process directly on the control computer (As the signal had to pass through the JAUS communication protocol first). Connecting a new camera to the laptop helped the software react in real time (while the video feeds from the robot were still accessible). While the built in cameras were not used in this particular experimental paradigm, having the ability to interface directly with the on-board cameras allows future projects to take advantage of views that may be more difficult to introduce with external hardware. The robot arm has a small camera which gives a "dog's mouth" perspective on the gripper. Having an accessible video feed from this camera will allow for much more complex and delicate manipulation in future tasks.

To facilitate planning of arm movements, we developed a simulation environment for the robot joints (Figure 2). The simulation environment is written as an extension of a 3D visualization environment that was designed for the ADE infrastructure. Code is written in native Java and can be run with the visualization on or off. This allows the simulation to be run on either a standard PC with a GUI showing progress or a high performance computational grid to run detailed large-scale searches of the complex arm movement space.

In the simulation environment, a robot arm is specified as a series of rigid blocks connected by joints. Blocks are attached to each other in a familial hierarchy starting with the root block that is connected to the robot frame. Each block is defined as a collection of primitive objects (spheres, rectangular solids, cylinders, etc.) with affine transformations applied relative to that block's starting position and orientation. As with most 3D tools, sub-objects can be nested in a parent block and an affine transformation specified for the entire block of sub-objects. Each joint is also capable of defining how that joint is allowed to rotate. Rotation can be in any of the three cardinal planes (around the X, Y, and Z axis). A center of rotation is specified both for block that is to be rotated as well as the parent block to which it is attached. A starting rotation angle can be given as well as bounds on rotation in each of the planes. All specifications are stored and read from an XML formatted data file.

When the control architecture determines a goal point to move the arm to, the planning mechanism in the simulation environment begins. A goal is specified as a coordinate in three-dimensional space relative to the root of the robot arm as well as an orientation of the end effector at that point. Planning is carried out using a gradient descent technique which can either be mimicked in real-time by the physical robot (and feedback used from the arm pose of the actual robot if available), or stored as an action sequence and played back at a future time. In the gradient descent algorithm, the robot attempts to move each of its joints a fixed small distance in each direction that the joint is allowed to move in. A fitness function is used to calculate the fitness of each of these new arm locations and the movement with the highest fitness is selected to be performed. This pattern is iterated until a local maximum is found. The fitness function is a scaled combination of the distance of the effector from the goal, the orientation of the effector relative to the desired end orientation and a negative strength from all joints based on their proximity to any collisions in the environment.

While gradient descent will not be sufficient to solve all arm trajectory problems, it is sufficient for many purposes. The M2P2 arm moves in a single vertical plane, with two joints controlling the position of the end effector. Since it also provides real-time (and can be pre-calculated in far faster than real-time) movements, it is ideal for the immediate response time required by tasks such as bomb defusing for which the M2P2 was designed. The simulation environment can also be used by more complex path planning algorithms for more complex environments.

One important complication to arm movement was the lack of "somatosensory" feedback, as there were no sensors to detect the relative arm position. To calculate the relative arm position, we took the arm resting pose as a base (this location was enforced through physical stops on the robot joints). Through experimentation, it was determined that the acceleration time in getting the robot arm from rest to a desired velocity was minimal. The amount of time necessary to move the arm a desired distance could be calculated simply by dividing the distance by the presumed constant speed of the arm. While this works for low fidelity situations, the acceleration will cause problems where fine motor control is required (such as using the grippers to pick up a small object). To reduce the error, all movements were blocked into constant length (or angle) chunks. As the gradient descent approach expected output tests to be constant sized already, constant sized movement blocks was the obvious choice.

If the robot had needed to perform multiple tasks, the cumulative errors would have added up. To combat this, the forced resting pose could be required after a certain amount of movement to recalibrate location.

3. Validating the Integration

To evaluate the integration, we devised a simplified version of the robot's primary function: bomb disposal. In this test, the robot was required to detect a suspicious object ("the bomb"), approach it, and retrieved it. It was then required to detect the ("safe") disposal receptacle, move to it, and deposit the "bomb" inside. This task can be decomposed into five phases, as depicted in Figure 3.

Detect object: Object detection was performed using simple color detection and blob size thresholding. The ADE vision component processed frames from the web camera, producing a description of each detected object that included its relative position in the frame (and hence relative to the robot's heading) and its size in the visual field. The autonomous control system made regular requests to the vision component until a positive identification was made, at which point control progressed to the next phase.

Move to object: Once the object was detected, the control module computed the course adjustments to move to it using a visual servoing approach [15]. Rotational adjustments were made based on the horizontal position of the object in the frame. Given the assumption that the object in question would be on the floor, distance could be estimated using the object's vertical position in the frame. When the object was determined to be in the effective region of the gripper, the approach phase was concluded.

Plan/perform reach: The reach and grasp actions were constructed dynamically using the arm motion planner described above. The "reach" goal was set based on the object's relative location as determined by the vision component. The control system sent the plan steps generated by the gradient descent algorithm to the arm, moving the gripper toward the goal state: poised over the target object.

Detect receptacle: The receptacle was detected in the same way that the target object was detected. A different color was used as the indicator.

Move to receptacle / drop: When the robot moved such that it judged its manipulator was over the receptacle, the block was released and the robot ended its movement pattern.

A video of the bomb disposal demonstration can be viewed at `http://tiny.cc/autonomized`.

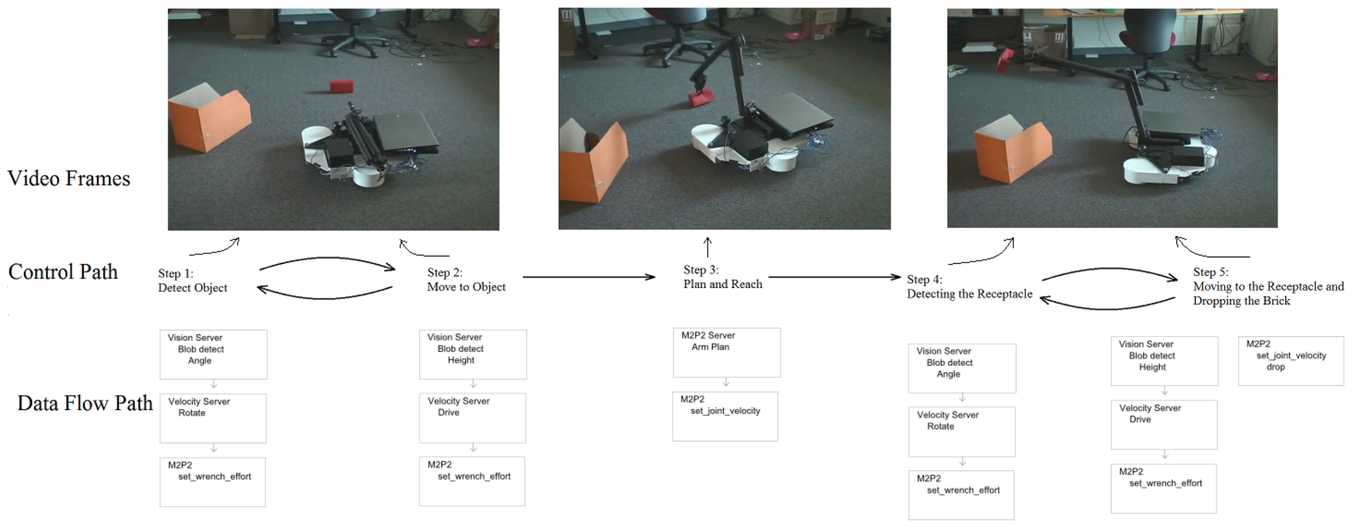

Fig. 3: **The data control path and data flow during the experiment (the control path indicates at what steps different computational units were activated).**

4. Conclusions and Future Work

Limitations to autonomizing teleoperated robots are dictated by the design of the robot in question and the persistence of the group attempting to perform the automation. In general, if basic platform movement can be achieved, most traditional robotic tasks can be performed with the possible addition of necessary sensors and manipulators. To this end, the basic platform can be thought of as an operation module with the potential addition of pre-existing functional modules attached. If very fine control is needed over the robots pose, it may not be feasible to augment a teleoperated system, since such fine feedback may not be available, and extremely difficult to add into the physical architecture. Although we used multiple software infrastructures in our case study integration, we used only one hardware platform (the M2P2 robot). In future work, integration of multiple existing hardware platforms would lead to a more modularized schema for conversion (e.g., a chassis from one manufacturer and a manipulator from another could both be integrated into the same software architecture).

References

[1] M. Luimula, K. Saaskilahti, T. Partala, S. Pieska, J. Alaspaa, and A. Lof, “Improving the remote control of a mobile robot using positioning and ubiquitous techniques,” in *Proceedings of the 2007 IEEE International Conference on Automation Science and Engineering*, September 2007, pp. 1027–1033.

[2] I. Farkhatdinov and J.-H. Ryu, “Hybrid position-position and position-speed command strategy for the bilateral teleoperation of a mobile robot,” in *Proceedings of the 2007 International Conference on Control, Automation and Systems*, October 2007, pp. 2442–2447.

[3] C. W. Nielsen, M. A. Goodrich, and R. W. Ricks, “Ecological interfaces for improving mobile robot teleoperation,” *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 927–941, October 2007.

[4] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. G. Chafin, W. C. Dias, and P. F. Maldague, “MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission,” *IEEE Intelligent Systems*, vol. 19, no. 1, pp. 8–12, January-February 2004.

[5] M. Goodrich, D. O. Jr., J. Crandall, and T. Palmer, “Experiments in adjustable autonomy,” in *Proceedings of the 2001 IJCAI Workshop on Autonomy, Delegation and Control: Interacting with Intelligent Agents*, Seattle, WA, August 2001, pp. 1624–1629.

[6] M. Y. Cheng and R. Cohen, “A hybrid transfer of control model for adjustable autonomy multiagent systems,” in *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, The Netherlands, 2005.

[7] P. Scerri, D. Pynadath, and M. Tambe, “Why the elf acted autonomously: Towards a theory of adjustable autonomy,” in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, 2002.

[8] R. D. Barreto, “Migration from teleoperation to autonomy via modular sensor and mobility bricks,” Master’s thesis, The University of Tennessee, Knoxville, August 2006.

[9] D. Stewart and P. Khosla, “Rapid development of robotic applications using component-based real-time software,” in *International Conference on Intelligent Robots and Systems-Volume 1*, August 1995.

[10] R. Cantrell, M. Scheutz, P. Schermerhorn, and X. Wu, “Robust spoken instruction understanding for HRI,” in *Proceedings of the 2010 Human-Robot Interaction Conference*, March 2010.

[11] S. Rowe and C. R. Wagner, “An introduction to the joint architecture for unmanned systems (JAUS),” Cybernet Systems Corporation, Ann Arbor, MI, Tech. Rep., 2008.

[12] J. Kramer and M. Scheutz, “Robotic development environments for autonomous mobile robots: A survey,” vol. 22, no. 2, pp. 101–132, 2007.

[13] M. Scheutz, “ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures,” *Applied Artificial Intelligence*, vol. 20, no. 4-5, pp. 275–304, 2006.

[14] V. Andronache and M. Scheutz, “Integrating theory and practice: The agent architecture framework APOC and its development environment ADE,” in *Proceedings of AAMAS 2004*. ACM Press, 2004, pp. 1014–1021.

[15] P. Corke, “Visual control of robot manipulators—a review,” in *Visual Servoing: Real-Time Control of Robot Manipulators Based on Visual Sensory Feedback*, K. Hashimoto, Ed. World Scientific, 1993, pp. 1–31.