Proceedings of Australasian Conference on Robotics and Automation, 2-4 Dec 2013, University of New South Wales, Sydney Australia

Goal Accomplishment Tracking for Automatic Supervision of Plan Execution

Nicolas Small, Graham Mann, Kevin Lee
Applied Artificial Intelligence Laboratory,
School of Engineering and Information Technology, Murdoch University,
Murdoch 6150, Western Australia, Australia
{n.small, g.mann, kevin.lee}@murdoch.edu.au

Abstract

It is common practice to break down plans into a series of goals or sub-goals in order to facilitate plan execution, thereby only burdening the individual agents responsible for their execution with small, easily achievable objectives at any one time, or providing a simple way of sharing these objectives amongst a group of these agents. Ensuring that plans are executed correctly is an essential part of any team management. To allow proper tracking of an agent's progress through a pre-planned set of goals, it is imperative to keep track of which of these goals have already been accomplished. This centralised approach is essential when the agent is part of a team of humans and/or robots, and goal accomplishment is not always being tracked at a low level. This paper presents a framework for an automated supervision system to keep track of changes in world states so as to chart progress through a pre-planned set of goals. An implementation of this framework on a mobile service robot is presented, and applied in an experiment which demonstrates its feasibility.

1 Introduction

All artificial systems are designed with a purpose in mind, whether accomplished by a single action, continuous process, or a series of subtasks. The principle of divide-and-conquer may be generally applied to subdivide these tasks until they reach a more manageable size. For example, in assembly lines-based manufacturing, the complex process of assembling a product is split into smaller steps. Scientific workflow processing aims to complete complex tasks by splitting them up into subtasks, utilising available distributed resources [Deelman *et al.*, 2004]. Software-based project management often splits the development process into smaller, more manageable units of activity [Kerzner, 1995]. In these examples, accomplishing a purpose is defined as successfully executing a series of sequential steps which alter the state of the world successively toward an ultimate goal.

In Automated Planning, a plan is represented as a graph in which the nodes are goals (the individual steps) and the actions are the links between them. In this paradigm, goals are defined as desirable states of the world [Ghallab *et al.*, 2004]. Achieving a goal amounts to choosing and then performing those actions from the agent's behaviours that are required to change the state of the world from an undesirable state to the desirable state specified by the goal. The plan is a tree of interdependent goals, with each parent goal relying on the fulfilment of all its child nodes. A task is accomplished when the root node (the highest-level goal) is satisfied. Such a plan presupposes a repertoire of world-altering actions and the ability to choose and execute these appropriately.

Monitoring the progress of subgoals is essential to managing plan execution. For example, the monitoring of a team during plan execution in business team management [Marks *et al.*, 2001]. This monitoring may be centralised, where information about each element of the team is collected in order to provide progress reports to each team member, or distributed among team members which both collect and share the information on a network. Teams can accomplish complex, dynamic tasks, such as in the 2006 surface team experiment at NASA's Ames Research Center, which studied a group of humans and robots performing planetary surface exploration tasks under the direction of an automated planner [Pedersen *et al.*, 2006]. In this case it is important that all team members understand which parts of the plan have been accomplished, which are still to be completed, and which are being currently undertaken[Perzanowski *et al.*, 1999].

In dynamic environments such as those encountered by mobile robots, plan execution needs runtime manage-

ment [Pedersen *et al.*, 2006]. This can be accomplished through a variety of approaches, such as workflow execution engines [Deelman *et al.*, 2004], adaptive schedulers [Lee *et al.*, 2009], and automated planning systems [Kephart and Chess, 2003]. Using these approaches allows for failure management through fault tolerance and task optimisation. These techniques require the tracking of goal accomplishment. This job can be performed centrally by assigning it to a single agent (autonomous or human), or it may be distributed by sharing it out between several agents of the team.

This paper proposes an architecture for observing agents and tracking the accomplishment of goals during the execution of a plan. It makes use of information available to it through an array of distributed sensors to test for world states that attest to goal satisfaction. The paper describes an experiment in which goal accomplishment by a teleoperated robot is supervised automatically using sensors at the worksite. These can be located on the teleoperated robot (as in the experiment), fixed at the worksite, or carried by a human worker; but are most likely to be located on the robot system, since these are needed there for other reasons [Kephart and Chess, 2003]. The remainder of this paper is structured as follows. Section 2 introduces the domain of manual and automated goal accomplishment tracking. Section 3 proposes an architecture for automatic goal accomplishment tracking. Section 4 evaluates the architecture using a small mobile robot performing an industrial maintenance task, while Section 5 briefly notes that the architecture can be used to facilitate machine learning by demonstration from a skilled human expert. Finally, Section 6 presents some conclusions and directions for future work.

2 Goal Accomplishment Tracking

Interest in the monitoring of the execution of plans is focused on supporting the improvement of this execution, and the improvement of the monitoring itself [Lee *et al.*, 2009]. These improvements are initially motivated by the specific domains they originate from, but tend to be generalised later if the ideas are not domain-specific. For example, in the business domain, languages such as BPEL (Business Process Execution Language), seek to describe the execution of business processes in order to provide some process management capabilities, such as lifecycle management, failure recovery, and a variety of control regimes, while mostly ignoring the data being transferred between those processes [Frank Leymann *et al.*,]. Scientific workflow processing, such as is supported by workflow management engines [Deelman *et al.*, 2004] is driven by the need to operate on large data sets [Sonntag *et al.*, 2010], therefore placing more importance on the data flowing between processes. Regardless of low-level differences, these systems provide similar high-level plan execution monitoring functionality, and seek to solve similar problems at that level.

To track the progress of a plan's execution, a monitoring system needs to be able to actively fetch or wait for an update on goal statuses during execution. BPEL does this through web service interfaces, whilst scientific workflow management systems utilise log file parsing. After the data is collected, it needs to be analysed for patterns, simple and complex, that indicate the current status of plan execution. Depending on the state of the plan execution, it can continue, or be re-planned. This process has been formalised by the Autonomic Computing community to support adaptive systems [Kephart and Chess, 2003].

It is convenient to represent goals in the same form as observed states of the world as provided by sensors. Checking the accomplishment of a goal might then be as simple as comparing it with a current set of world-state representations. For example, if a robot is equipped with a sensor which returns its position in two-dimensional space, a locational goal might be specified as 'robot_at(250, 300)'. This can be directly compared to the output of the sensor, which might return 'robot_at(240, 300)'. A more sophisticated arrangement would involve abstracting the goal to 'robot_at(waypoint)' and providing additional knowledge defining the waypoint in terms of a number of sensory tests and satisfaction condition with respect to those tests. In this case, the waypoint is associated with a set of GPS coordinates of an area within which the target is found, a specific barcode known to be present at the target an image pattern to be matched against a known landmark through the use of an on-board camera. The satisfaction condition is that the current GPS coordinates must be within bounds, and that a match on either of the other sensory indicators would be sufficient. Such straightforward arrangements would not always suffice, because the relationship between sensory data and actual world states is not always simple. Not only does the sensitivity, range and signal-to-noise ratio characteristics of a given sensor affect the interpretation of its signals, but the satisfaction or failure of some goals might involve a subtle alteration in sensed properties, possibly including necessary state progressions or alternatives. Some of the literature on robot perception deals with the control of uncertainty introduced by these complications [Thrun, 2000] [Minguez and Montano, 2005].

Futhermore, not all goals are the same, but may have different natures based on their objective and relation to processing. In this paper we distinguish three types of goals:

- **Achieve goals** [Dastani *et al.*, 2006] are simple expressions denoting a desired world state to be

reached.

- **Maintain goals** [Dastani *et al.*, 2006] are goals that need to be protected (i.e. their accomplishment tests must not be allowed to fail). Maintenance goals require extra monitoring after they have been initially accomplished, placing an extra burden on the monitoring system.
- **Opportunistic goals** are goals associated with a watch for particular events or world-states, the presence of which is considered favourable. Opportunistic goals mirror maintain goals, in that rather than demand checks for threats to goals, they encourage checks for contingencies favourable to goal accomplishment.

These goal types require different monitoring support. Achieve goals depend upon matching the required world-state to the current state of the world. Maintain goals seek to actively protect a desired state. This requires tests sensitive to boundary conditions around the goal state which suggest a threat, requiring protective actions to be executed. Such protective actions can be included the hierarchical plan graph as special annotations. Opportunistic goals must use tests to detect occurrences known to promote the accomplishment of goals, as well as the appropriate actions which may be similarly included in the plan.

3 An Architecture for Automatic Goal Accomplishment Tracking

This section proposes an architecture to provide goal accomplishment tracking to support the situations described in Section 2. This architecture is designed to remove the need for agents to report progress at every step, shifting this role back onto an analysis module in charge of the tracking of plan execution progress. This module monitors the world, using data provided by sensors, for the changes expected to occur when each goal has been accomplished. The module then updates a central plan-tracking structure with the progress made (See Figure 2).

This approach is centralised by design, to account for the varying range of capabilities displayed by agents in various domains. While some automated agents, such as automated robots, might do goal accomplishment tracking internally to ensure proper execution of their assigned tasks, other more basic agents simply encounter a situation and act in response without ever checking to see if their goal is accomplished. In the case of human or human-controlled agents, goal accomplishment is understood by the human in question but has no easy way of being broadcast back to other agents without additional tools. As soon as any of these agents are to work together, a centralised approach to goal accomplishment tracking is required.

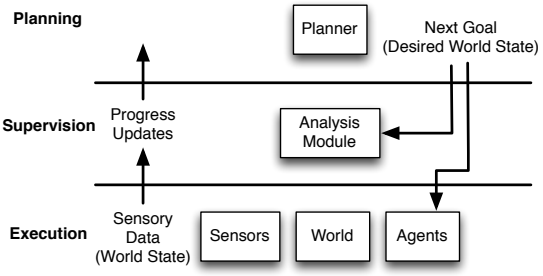


Figure 1: The Architecture

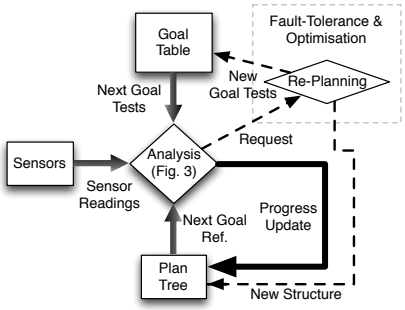


Figure 2: The Supervision Process: Data Flow

3.1 Requirements

The architecture assumes the presence of several elements (See Figure 1), without which it has neither the means nor the reason to fulfil its purpose.

- **A plan** to provide both a reason for the architecture to be in place, and the framework for the analysis module to operate within. Without a plan, agents cannot know what changes to make to the world state, and the analysis module cannot know what changes to expect.
- **One or more agents** to effect changes on the world. Without them, the plan cannot be executed, and there exists no state changes to track.
- **Sensors** to allow the reading in of the state of the world. Without them any changes caused by the agents cannot be observed and tracked.

3.2 Runtime Activity

During runtime, the architecture feeds sensor inputs through the analysis module, to perform relevant tests (See Figure 2) and executes any additional functions required of it (See Section 3.3).

Inputs

For its most basic functions, the analysis module only requires three types of inputs: the plan's structure, de-

sirable world states and actual world states.

- **The Plan Tree** is the name given to the data structure that holds the logical information about the goals. This includes the overall hierarchy and the relationship of the goals to one another, and goal status (i.e. achieved, in progress, not started, and maintaining).
- **Desirable World States** are generated during the preparation step (See Section 3.4), these states are fetched by the supervisor from a data storage element, such as a lookup table.
- **Actual World States** consist of the information reported by the sensors queried by the architecture.

Analysis

Processing the sensor data during runtime can be accomplished using a relatively simple algorithm. An example solution is presented in Figure 3. This example iterates through the goal list, checking for each goal (i) the availability of the required sensors, and (ii) whether or not the sensor data matches the desired results.

```
for each goal G_i do
    flag ← TRUE
    for each sensor S_k in satisfaction_conditions (G_i) do
        if needed(S_k, G_i) then
            if power_up(S_k) then
                wait(max.sensor.powerup.time)
            end if
            if inactive(S_k) then
                message: "Can't test G_i, sensor S_k is inactive"
                flag ← FALSE
                break
            else
                if ¬ match(S_k[W_l], S_k[W'_l]) then
                    flag ← FALSE
                    break
                end if
            end if
        end if
    end for
    if flag ≡ TRUE then
        message: "Goal G_i is satisfied"
        leave_on(G_{i+1})
    end if
end for
```

Figure 3: Example Algorithm Testing for Goal Satisfaction, Including Sensor Power Management

The algorithm calls on several functions explained below:

- **needed(S_k, G_i)** returns false if i) the satisfaction conditions of G_i contain an expression in disjunction with S_k, and ii) that expression currently evaluates to true; true otherwise.
- **power_up(S_k)** checks to see if S_k needs powering up. If so, powers up S_k and returns true; false otherwise.
- **inactive(S_k)** returns true if sensor S_k is not currently returning a valid world observation; false otherwise.
- **match($S[W], S[W]$)** returns true if world observation W agrees with corresponding goal observation W'; false otherwise.
- **leave_on(G_{i+1})** checks if the next goal (G_{i+1}) requires any sensors already used by G_1; all others are powered down.

Note that this version of the algorithm does not distinguish between the three types of goals. To do that, the algorithm of Figure 3 would need to be modified to provide special processing for maintenance goals and for opportunistic goals. Maintenance goals would need extra guard tests and a stack of monitoring processes. Opportunistic goals need equivalent extra tests for favourable circumstances with the associated stack of monitoring processes. Once a goal was processed, these processes would run continually until the algorithm terminated.

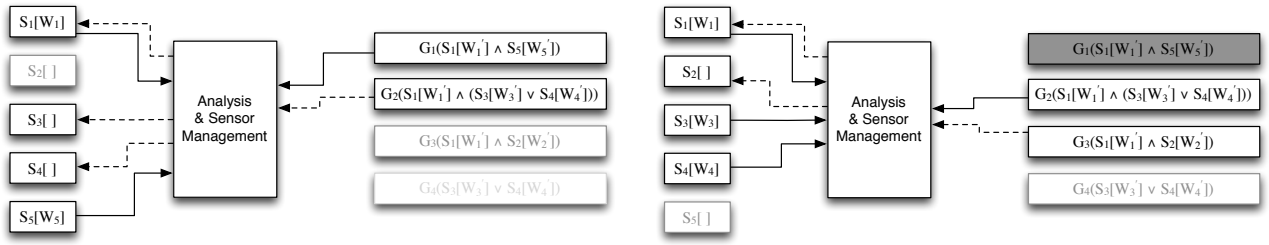
Output

To serve as a monitoring entity, the architecture needs to be able to send messages. In its most basic form, the architecture will post progress updates on the plan data structure. This structure is centralised to allow updates to be rapidly propagated to each other elements of the system; for example, agents can periodically query this central structure to get an up-to-date picture of the plan's completion status. The number of outputs will vary based on individual implementations and their needs for extra functionality. For example, a system requiring some fault tolerance (See section 3.3), will need to be able to request some form of re-planning.

Sensor Management

Managing the fusion of data from multiple sensors to test for a single condition, and dealing with the number of different sensors required by the architecture to be useful, both create issues that must be overcome for the architecture to be successful. The sensor fusion problem needs to be considered for each each goal if the accomplishment tracking for that goal relies on the input of multiple sensors. One simple solution is to express the relationship between the sensors as a set of boolean functions as shown in Figure 4. The specification of how sensory tests apply to each goal is performed during the preparation step (see Section 3.4).

The advantage of knowing which sensors will be needed for each goal is that predictions about sensor uses can be made. This enables us to: (i) enact power management policies (sensors can be turned on only



(a) Accomplishment tracking of goal G1, and sensor readying for goal G2

(b) Accomplishment tracking of goal G2, and sensor readying for goal G3

Figure 4: Goal Accomplishment Tracking and Sensor Management

when needed, and off when not), (ii) ensure sensors are turned on early to ensure any sensor startup delays are accounted for, and (iii) check sensors for malfunctions ahead of use, leaving time for any potential re-planning to be made. This sensor management is illustrated in Figure 4, with Figure 4a showing a possible state for the system to be in when supervising the accomplishment of a goal G_1. G_1 requires tests on data from sensors S_1 and S_5, providing world states W_1 and W_5 respectively. These will be compared to desired world states W_1' and W_5'. The system is also pre-emptively preparing sensors S_1, S_3, and S_4, which are required by goal G_2. Figure 4b shows the state of the system after goal G_1 is accomplished. The accomplishment of goal G_2 is being supervised while sensors required by the monitoring of goal G_3 are being prepared. This kind of management is especially important for mobile robots, because of they typically have limited power and computing resources.

3.3 Fault Tolerance and Optimisation

A fault tolerant system has the ability to respond to any event that might cause a problem at runtime [Randell, 1975]. Optimisation attempts to take advantage of opportunities to improve the execution of plans during runtime [Lee *et al.*, 2009]. Both fault tolerance and optimisation are a feature of many current workflow monitoring systems. Proposing approaches to dealing with these processes is beyond the scope of this paper; however, facilitating the inclusion of such capabilities in this architecture will broaden its usefulness.

Both fault-tolerance and optimisation require some monitoring of the processes occurring in the system, making the proposed architecture suitable to take on the role of notifier. Any condition requiring fault-tolerant or optimising processes to engage could be specified as a maintenance (or opportunistic) goal in the overall hierarchy, the realisation of which would trigger those processes. Figure 2 shows how any re-planning needed by these processes could be kept distinct from the analysis module, only having an impact on the inputs and outputs used by the analysis.

3.4 Preparation

The architecture presented here is applicable to a wide range of scenarios, and as such is designed in a high-level and modular fashion. To prepare the automatic goal execution tracking for a particular application(industrial maintenance by robot), the following steps have to be performed:

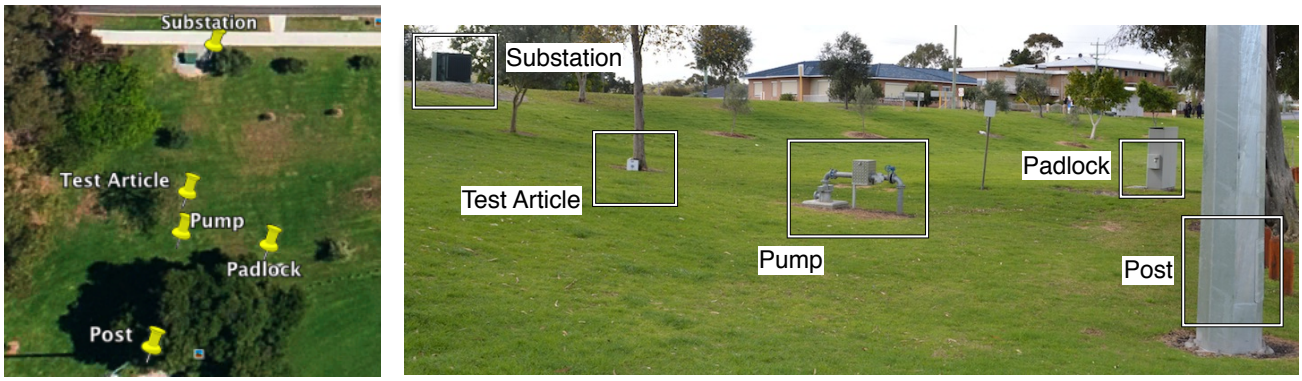
1. **Enumerate available sensors**. Sensors available to the architecture must be found and recorded in a list. It is necessary to include in the list whether or not the sensors are bound to a particular location or are mounted on a mobile platform.
2. **Extract goal list**. Before a series of tests for each goal in the plan can be made, a list of those goals must be drawn up.
3. **Match goals with sensors**. The two lists must be cross-referenced to associate each goal with at least one sensor. This match up will depend on several factors: (i) the ability of the sensor to produce meaningful information about the goal's status, and (ii) the location and mobility of the sensor (i.e. can the sensor 'sense' at the right location).
4. **Convert abstract goals to usable world states**. Each goal must be converted from an abstract meaning to usable boundaries with which sensor data can be classified. This forms our desirable world states.

This process is demonstrated through an experiment in the next section, where the architecture is tested with real data collected by mobile robots sensors in the field.

4 Experimental Evaluation

4.1 Overview

This goal-tracking architecture is an essential component to an "Assigned Responsibility" teleoperation framework



(a) Reference Coordinates Plotted in Google Earth

(b) Photograph of the Test Site

Figure 5: The Test Site and Points of Interest

currently being developed by the authors. This framework allows the responsibility for the execution of individual subgoals in a plan to be explicitly assigned at configuration time to specific operators (usually a human operator or an automated operator) who accomplish the subgoal by controlling a robot. Managing the change of operator between subgoals requires a clear understanding of what has been accomplished previously, to both trigger the change, and to provide context to the operator beginning their time in control. This tracking must be done independently of the operators to ensure continuity in the tracking regardless of operator capabilities, by an analysis module (see Section 3) embedded in the "Assigned Responsibility" interface.

To validate the goal accomplishment tracking architecture, an experimental approach has been chosen using a mobile field-robot and a series of simple tasks. The first experiment consists of a baseline validation of the sensors chosen when applying the preparation steps presented in Section 3.4. A second experiment evaluates the ability of the chosen sensors to detect the successful accomplishment of the goals in the field.

4.2 The Maintenance Photography Task

Automatic regular maintenance of physical equipment requires a mobile robot to periodically visit a number of key worksites, where the robot may perform tasks such as photography, gathering sensor data on environmental conditions, physically probe the integrity of surfaces, joints or attachments, remove panels and/or to change out faulty components [Mann, 2008].

In this experimental scenario, a robot is teleoperated around a series of five worksites (See Figure 5), aligning itself close to each one in turn so as to be able to photograph important objects. The goals of being at each photograph point are defined by a bounding circle of GPS coordinates associated with the worksite and an image for matching using the vision system. A goal is met only if the robot's current GPS coordinates fall within the specified range and the vision system is suitably confident of a matching visual image. The remainder of this section demonstrates the application of the automated supervision system to this scenario.

4.3 The Mobile Field Robot Agent

The experiment uses a Coroware mobile field robot, built on top of a Lynxmotion base with four driven wheels, with an Intel D2700MUD Mini-ITX single-board computer running Linux. In contains a steerable camera, a modified 5 DoF manipulator arm (Figure 6) and is equipped with a variety of sensors as described below. It is linked via 802.11g wireless to a laptop and Logitech game controller that serve as a teleoperation station.

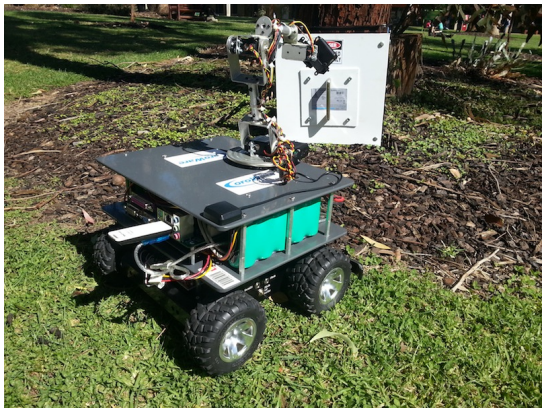


Figure 6: The Coroware Mobile Robot

The robot and the teleoperation station run custom Python-based code allowing commands to be sent from the teleoperator to the robot and translated to low-level motor commands. Sensor readings from the robot are

transmitted back to the operator for processing on the teleoperation station.

4.4 Experiment One: Sensor Testing

Overview

The robot supports a number of sensors typical for mobiles of this kind, including bump switches for obstacle avoidance, battery level sensing, potentiometers sensing angles on the manipulator arm and a force sensitive resistor measuring gripper pressure. For the purpose of this experiment, the important sensors include a global positioning system (GPS) module and a steerable video camera. Position, velocity and direction data updates are obtained from a PhidgetGPS board with a best-case circular error probability of 2.5m. Forward imagery for teleoperation control is obtained through a front-mounted 1600 x 1200 pixel Logitech QuickCam Orbit AF camera, which can be panned under software control through 189 degrees and tilted through 102 degrees.

Pattern recognition of landmarks and task states is achieved by a Speeded-Up Robust Features (SURF) [Bay *et al.*, 2008] module from the Linux OpenCV library. This is a scale and rotation invariant image matching algorithm, which can be used to compare the camera input to a reference image based on correspondences of interest points in the two sources. We have found this algorithm to quite reliable in the face of visual disturbances such as changes in lighting due to the weather and interposing objects such as hands, tools etc. To understand how the sensors behave in field conditions and to develop the goal satisfaction tests, baseline measurements for those sensors were first recorded.

Results

The GPS sensor was tested in a best-case scenario, in an open park under clear skies. Seven series of roughly one minute each, recorded over a period of three days from the same location are plotted in Figure 7. This totalled 1059 individual readings, which were found to diverge on average from a calculated mean position by 3.53 meters. As Figure 7a shows however, some of these readings diverge by up to 40 meters. Most of these divergences occur early in the sample, as the calculated location tends to converge towards the mean position after a few seconds, but some occur later on in the sample as, presumably, the sensor loses contact with one or more satellites. These eccentric readings tend to be in the minority, and can be mitigated by calculating a mean position using several seconds of readings.

These results motivated the choice of an appropriate difference threshold of four meters as the criteria for success in the GPS test. This four meter radius circle is plotted in Figure 7. Figure 7b shows the readings contained within the bounding circle.

Hessian T./ nOctaves	300		400		500	
	Target	No Target	Target	No Target	Target	No Target
2	21.89	3.943	20.78	4.525	19.71	5.15
3	19.24	5.257	23.16	4.33	19.78	4.46
4	21.91	3.567	24.33	5.8	19.33	3.47
5	24.19	3.767	25.46	4.237	17.63	3.34
6	23.26	3.607	23.89	4.81	20.73	5.11

(a) Key Points Detected for Each nOctave / Hessian Threshold Combination

Hessian T./ nOctaves	300	400	500
2	5.55	4.59	3.83
3	3.66	5.35	4.43
4	6.14	4.19	5.57
5	6.42	6.01	5.28
6	**6.45**	4.97	4.06

(b) Signal-to-noise Ratio for Each nOctave / Hessian Threshold Combination

Table 1: Results of the SURF Algorithm Tests

The behaviour of SURF was studied in the laboratory under systematic adjustments to two key parameters, nOctaves and hessianThreshold, in order to optimise its performance. nOctaves sets the number of Gaussian pyramid octaves used for matching, which affects the grain size of detected features. hessianThreshold sets the acceptable feature strength for retention of matched interest points.

The algorithm was modified to return a confidence value expressing the number of matched interest points in real time. Table 1 shows number of observed matched points and signal-to-noise ratios for 2-6 octaves and Hessian thresholds over the recommended range of 300-500. We chose nOctaves of 6 and a hessianThreshold of 300 because these returned the best signal-to-noise ratio and a reasonable number of points on our test target object.

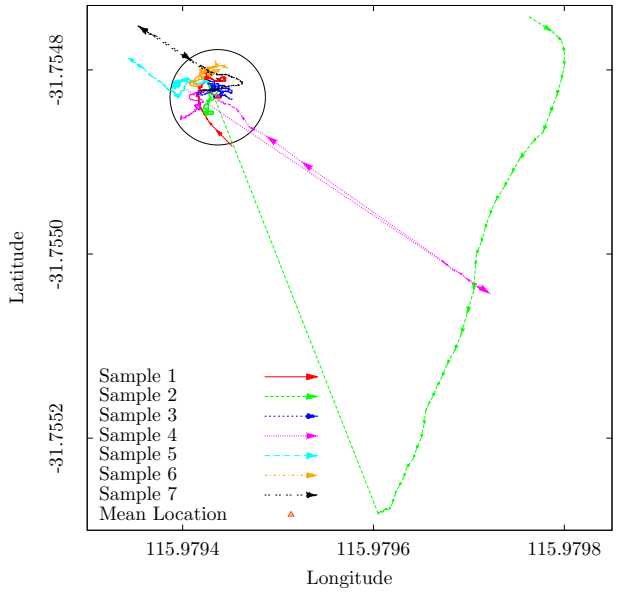
4.5 Experiment Two: Field Testing

Overview

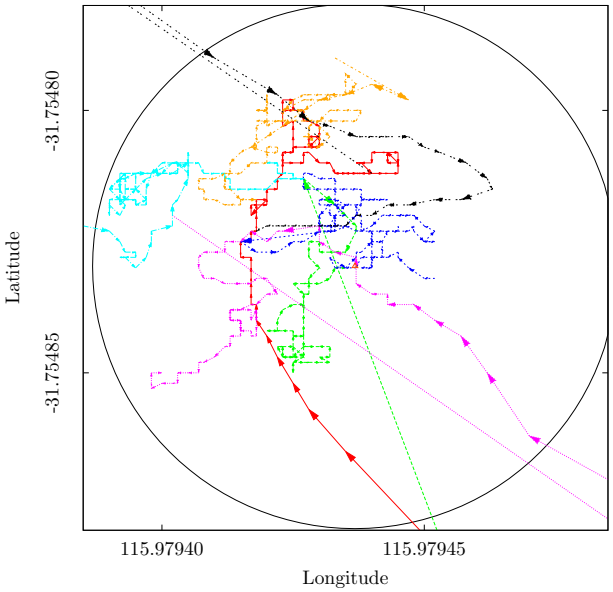
The experimental scenario (See Section 4.2) describes a task where a robot is required to navigate around a work site and take photographs of five landmarks. This experiment was designed to verify that sensor readings could be used to confirm the accomplishment of the goals detailed in the plan. The sensors and processing software tested in Section 4.4 were mounted on the Corobot (Described in Section 4.3) which was then driven around the test site (Pictured in Figure 5) while its presence at the required location was confirmed by the sensor tests.

Results

Readings from the GPS sensor were taken at each desired position, and the distance of these readings from the desired coordinates was computed using the haversine formula. The recorded distance is plotted over time in



(a) Overall View of the Points Collected



(b) Zoomed-in View of the Area Directly Surrounding the Mean Coordinate

Figure 7: Baseline Readings Using the GPS Sensor. Readings Taken in Seven Groups Over Three Days in Ideal Conditions. A Four Meter Radius Circle was Centred on the Mean Coordinate

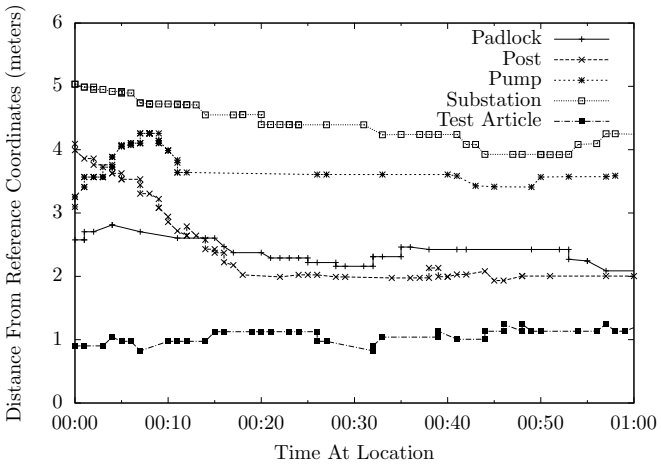


Figure 8: Recorded Distance from the Reference GPS Coordinates over Time

Figure 8. When the antenna of this device returns to a marked position, its behaviour showed a characteristic settling over a period of 60 seconds, as the position error fell to a low of about 4m at worst. This level of accuracy could probably be improved, but was sufficient for our purpose.

The sensitivity of SURF to variations in viewing angle in the horizontal was evaluated by recording the average number of matched interest points over a range of angular displacements of the robot with respect to the normal. Figure 9 profiles these counts for each target object in our field experiment, together with a non-target count taken by pointing the robot in a random direction away from each object.

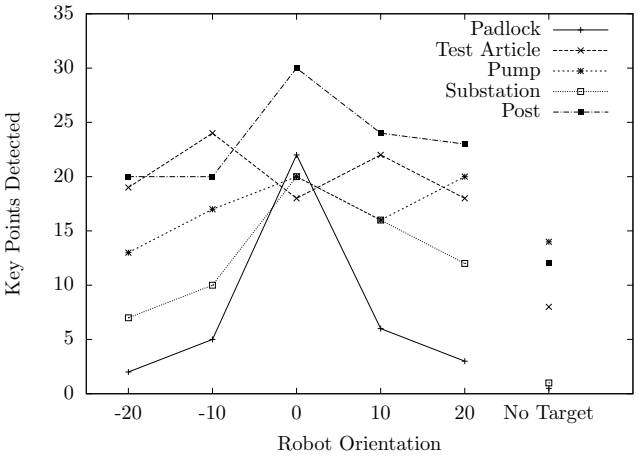


Figure 9: Key Points Detected by the SURF Algorithm at Each Waypoint

The number of matched points tended to decline systematically as angular displacement increased. For the

purposes of placing a camera for maintenance photography, it would be better if the range of acceptable angles satisfying the location goal was not too large. The results show that by setting a confidence threshold of approximately 20, the robot could almost always distinguish between target and non-target objects when the robot was in the correct position for photography, i.e. approximately normal to the object of interest.

The Boolean conditions of each subgoal (being at one of the five photography points), indicate that the current GPS coordinates fell within a bounding circle of radius 4m centred near each target object and that the SURF recogniser returned a sufficiently confident indication that the target object had been acquired. In all and only those times when the robot was in a suitable position for the photograph, this returned true.

Other goal states can be detected using these sensors. The object designated 'Test Article' in Figure 6 has a panel with a hatch from which the robot might be required to unscrew four bolts, remove it to gain access, withdraw a faulty part, replace it with a working part, then replace the hatch and secure it. A test of the subgoal of the hatch being removed was performed by training the SURF algorithm to recognise an open hatchway. Under a similar range of angles and visual disturbances, the tests reliably recognised the state of the hatch; open or closed. Note that these changes would be detected whether a human or robot agency had made them.

5 Supporting Learning by Demonstration

The teleoperation experiments in our laboratory require examples of automated controllers to be mixed with human control. One of the benefits of teleoperation is the opportunity to learn skilled action from a human expert, delivered directly to the machine in real contexts [Mann and Small, 2012]. Learning by Demonstration (LbD) is an approach involving the extraction of a robot control policy from a set of demonstrations of the target skill, recorded from the sensors and motor outputs (for a review of these approaches see [Argall *et al.*, 2009]).

Goal accomplishment tracking is important in this context because it can be used to partition large demonstration datasets containing multiple, possibly very different functional relationships into manageable, more easily learned sub-units. For example, Isaac & Sammut [Isaac and Sammut, 2003] were able to use human demonstrations of basic flying manoeuvres to learn compact, robust and transparent controllers ('behavioural clones') which both discovered how the pilots were setting goals and then learned how to cause the simulated aircraft to meet those goals. By learning how to set goals, and then learning how to control a dynamic system to reach those goals, the cloned behaviour can be learned more easily and show much greater robustness to changed conditions or plans than when learned from undifferentiated demonstrations of an entire task. We believe that automatic goal monitoring is essential to switch between these learned modular units of behaviour during the execution of complex tasks in dynamic environments, whether by human or robotic agency.

6 Conclusion

This paper proposes an architecture for monitoring a plan's execution. By placing an automated goal accomplishment tracking module at the center of a plan execution system, it is possible to combine data provided by a variety of sensors to provide accurate tracking of each goal's accomplishment in the plan. This supervision approach removes the need for agents to update a plan with their progress, and the centralised nature of the process allows for the rapid dissemination of plan progress information. The application of the architecture was demonstrated using an example maintenance task as executed by a mobile robot. The architecture is modular in nature, and well-suited for supporting often-needed functionality in the planning domain including fault-tolerance and optimisation, in addition to providing automatic segmentation to information used in learning by demonstration techniques.

Future work will implement of this architecture in a context where both human and machine agencies will share control of a single mobile robot. The ability to successfully track the accomplishment of goals will be crucial in this sliding automation environment where control responsibility must be clear at all times to avoid conflict.

References

[Argall *et al.*, 2009] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, May 2009.

[Bay *et al.*, 2008] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[Dastani *et al.*, 2006] Mehdi Dastani, M. Birna Van Riemsdijk, and John-jules Ch Meyer. Goal types in agent programming. In *In Proceedings of the 17th European Conference on Artificial Intelligence*, pages 220–224, 2006.

[Deelman *et al.*, 2004] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, page 1120, 2004.

[Frank Leymann *et al.*,] Frank Leymann, Dieter Roller, and Satish Thatte. Goals of the BPEL4WS specification. `http://www.oasis-open.org/committees/download.php/3249/Original%20Design%20Goals%20for%20the%20BPEL4WS%20Specification.doc)`.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, May 2004.

[Isaac and Sammut, 2003] Andrew Isaac and Claude Sammut. Goal-directed learning to fly. In *Twentieth International Conference on Machine Learning (ICML-2003)*, page 258265, 2003.

[Kephart and Chess, 2003] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[Kerzner, 1995] Harold Kerzner. Project management: a systems approach to planning, scheduling, and controlling. *New York,*, 1995.

[Lee *et al.*, 2009] Kevin Lee, Norman W. Paton, Rizos Sakellariou, Ewa Deelman, Alvaro AA Fernandes, and Gaurang Mehta. Adaptive workflow processing and execution in pegasus. *Concurrency and Computation: Practice and Experience*, 21(16):1965–1981, 2009.

[Mann and Small, 2012] Graham A. Mann and Nicolas Small. Opportunities for enhanced robot control along the adjustable autonomy scale. In *Human System Interactions (HSI), 2012 5th International Conference on*, pages 35–42, 2012.

[Mann, 2008] Graham A. Mann. Quantitative evaluation of human-robot options for maintenance tasks during analogue surface operations. In *Proceedings of the 8th Australian Mars Exploration Conference*, page 2634, 2008.

[Marks *et al.*, 2001] Michelle A. Marks, John E. Mathieu, and Stephen J. Zaccaro. A temporally based framework and taxonomy of team processes. *The Academy of Management Review*, 26(3):356, July 2001.

[Minguez and Montano, 2005] Javier Minguez and Luis Montano. Sensor-based robot motion generation in unknown, dynamic and troublesome scenarios. *Robotics and Autonomous Systems*, 52(4):290–311, 2005.

[Pedersen *et al.*, 2006] Liam Pedersen, William J. Clancey, Maarten Sierhuis, Nicola Muscettola, David E. Smith, David Lees, Kanna Rajan, Sailesh Ramakrishnan, Paul Tompkins, and Alonso Vera. Field demonstration of surface human-robotic exploration activity. In *AAAI Spring Symposium: Where no human-robot team has gone before*, 2006.

[Perzanowski *et al.*, 1999] D. Perzanowski, A. C. Schultz, W. Adams, and E. Marsh. Goal tracking in a natural language interface: Towards achieving adjustable autonomy. In *Computational Intelligence in Robotics and Automation, International Symposium on*, pages 208–213, 1999.

[Randell, 1975] Brian Randell. System structure for software fault tolerance. *Software Engineering, IEEE Transactions on*, (2):220–232, 1975.

[Sonntag *et al.*, 2010] Mirko Sonntag, Dimka Karastoyanova, and Ewa Deelman. Bridging the gap between business and scientific workflows: Humans in the loop of scientific workflows. In *e-Science (e-Science), 2010 IEEE Sixth International Conference on*, page 206213, 2010.

[Thrun, 2000] Sebastian Thrun. Probabilistic algorithms in robotics. *Ai Magazine*, 21(4):93, 2000.