An Action Pool Architecture for Multi-tasking Service Robots with Interdependent Resources

Tapio Taipalus and Aarne Halme

Abstract—We present a novel control architecture for execution of multiple tasks simultaneously. Traditionally, the task control architecture of a service robot has been focused on executing one task at a time. This has mainly been due to the history of mobility being the only function and tasks being so simple that just one or two abstract commands have been sufficient. During the last decade, these requirements have changed drastically and a more versatile operation is clearly required.

Service robots commonly have physical subsystems such as manipulators and directable perception sensors that are not in use all the time. These task execution resources are dependent on the pose of the platform to which they are attached. Our architecture manages the concurrent use of these resources in order to achieve parallel task execution.

We verify the architecture with experiments on two different robot platforms while performing concurrent tasks such as finding object, greeting people and taking pictures of a designated object. The proposed architecture can receive an arbitrary number of tasks to be executed at any time instant within limitations of computing resources.

I. INTRODUCTION

Our ultimate goal is to design a generic service robot capable of multi-tasking and assisting us in our daily activities. More specifically, we are focusing on mobile manipulator robots in an environment shared with humans. Complexity of the service robot in this context can not be reliably simulated, so we prefer experiments in real world scenarios. In order to be intuitively usable to a wider public, interaction with the service robot should mimic interaction with any other actor, human or animal, in our environment. Some basic features, such as autonomy, are expected. From control architecture’s point of view this means that you can give new tasks to the robot or take them back whenever you notice that something needs to be done. Naturally, the architecture should pursue these given tasks in parallel and concurrently, utilizing all available resources on the service robot.

For this challenge we see high level control as the greatest hurdle. A service robot requires a multitude of technologies to be integrated together. Most of these technologies, such as computation, mechanics and feedback control, are already mature enough to meet the needs of our design objectives but perception is still struggling to provide us with sufficient information about the surroundings. The artificial intelligence modeling our world is another technology whose realization is deemed to be even further away in the future.

Today’s service robots are well capable of executing multiple tasks sequentially. But, particularly in a shared environment with humans, many tasks have waiting times in middle of the task execution. During these waiting times, many of the expensive resources of service robots are either on standby or not utilized to their full extent. Our aim is to take advantage of the waiting times and utilize the unused resources of the robot, such as directable sensors, while they are not needed for high priority tasks. The same property is common in biology and makes behaviours of animals flexible and expedient.

A typical assumption in high level control architectures of service robots is that the robot would know the consequences of its actions and be able to predict the world state in a time span of several minutes or more. This is a valid assumption in isolated cases [1] but it falls short in shared environments. As a direct consequence, the task given to the robot has to be defined as a goal state. Planning algorithms are based on the decomposition of the goal to subgoals until the subgoals can be matched with the action-space of the service robot [2], [3]. This is how humans are known to pursue complex tasks. Robotic task control with these assumptions has been demonstrated in previous works but under the limitation that the preconditions and effects of a task are tailored to match the robot’s limited perception capabilities [4].

We are still very far from automating the process of deducing the needed actions to reach a goal. The robot would need to know not only the direct consequences of its actions but also to deduce indirect consequences and effects of a third party’s actions. Consider the scenario where a human or a robot is moving a glass filled with a liquid substance. The human would intuitively avoid spillage depending on the environment, urgency of the movement or value of the liquid, while the robot would have difficulty even noticing if it spills something.

In our proposed control architecture, we combine the strengths of the human brain and computing power. The human operator plans and sets the goals. The computer in turn, remembers a large set of detailed plans and objectively selects and executes the right plan based on its understanding of the current world state, of course can be limited. These plans are essentially one form of computer program with a lot of exception handling which makes them as reliable as possible. In our proposed architecture, exception handling is considered as a separate process from the so-called basic plan execution. This way, the system can concurrently react to exception with multiple tasks.

In our approach, we mix the order of execution for
multiple tasks so that parts of tasks which utilize the inter-
dependent resource of the robot can be executed in parallel.
This mixing of task execution is based on a novel way
of dividing the task into independent blocks. The proposed
architecture has some resemblance to software agent archi-
tectures. In fact, approaches inspired by Believes-Desires-
Intentions (BDI) agent paradigm [5], [6], [7] have many
similar mechanisms to ours. Our Action corresponds to the
Intention, our database corresponds to the Belief and our
task can be likened to the Desire. The major difference is
that instead of trying to automatically refine a goal from the
Desire and reason the Intention from the perceived world
state, we leave this up to the task creator, i.e. the human.

To demonstrate the operation of the architecture two
different service robot platforms are used. With the first
platform, an object is searched for and simultaneously a
picture taken of any human coming sufficiently close. With
the second platform, pictures of multiple objects are taken
and simultaneously people stopping by the robot are greeted.
Also the task evolvement is demonstrated in the picture
taking task by preventing humans from appearing in the
captured images. The usage of the interdependent resources
is demonstrated by concurrently looking for humans while
taking images of an object.

The rest of the paper is organized as follows. In the next
section, we describe the basics of the proposed architecture,
followed by two sections describing the experiments on
either platform. Finally, we discuss the results and future
work.

II. THE ARCHITECTURE

We divide the task into units called Actions and divide
the service robot to physical subsystems called Resources.
The Action is an atomic operation from the Resource's
point of view. A physical Resource occupies physical space
and can not be shared for multiple tasks in small fractions
like how computer processor can be used. In this context,
typical Resources in a service robot are the pan-tilt-unit,
the manipulators and the mobile base. Depending on the
application and tasks, the Resources can be further divided
e.g. to arm and gripper in manipulator. Our architecture
manages the usage of these interdependent Resources. We
consider Action as a reservation token for the Resource at
hand. Each Resource has its own pool of Actions and only
one Action from each pool can be under execution at any
given time. This way, the limited Resources can be scheduled
between tasks.

The famous three-layer control architecture model divides
architecture to reactive feedback layer, sequencing layer and
deliberative planning layer [8]. According to this terminol-
ogy we concentrate on the reactive sequencing layer. The
reactive feedback layer and a mechanism to sequence its
reactive functions is encased in a plan execution module. The
execution module is fed with a section of a task abstracted
into a unit called Plan. The Plan is defined as a set of
instructions to achieve some functionality or goal and it is
designed by an intelligent planner, i.e. the human. The Plan
is in turn encapsulated into the Action (Fig. 1). Action is
a function with a Resource that can not be divided without
compromising the success of the Plan.

With this division of Action and Plan we can modularize
the architecture and use any kind of implementation for the
reactive layer. In our implementation, the Plan is constructed
as a flowchart and translated into XML similar to our
previous work [9] but it could also be presented like [10]
or as a set of behaviours [4]. Before the Plan is executed the
Resource related to Action is reserved. For example, when
the robot's pose is a Resource, Plan is executed after the
robot has moved to the desired pose.

Our architecture is constructed from distributed software
components:
  • Action Pool (AP) combined with plan execution module
    for each resource
  • The current world state as a database
  • Event Listener (EL) agents that follow the world state
  • Perception agents that update the world state

![Fig. 1. An overview of the task structure](image)

![Fig. 2. An overview of concept in the action pool](image)

A. Action Pool

There is only one Action Pool (AP) to manage each
Resource of the service robot. AP selects the Action to use
the Resource based on Action's priority and expected total
execution time. Action selection can be done with different
policies in order to alter the behavior of the robot. Task is
assigned to one AP. Task is then executed by adding a set of Actions into the pool from task’s list. A new set of Actions is added when all the previously added Actions are executed. Fig. 2 explains this concept in more detail.

Resources managed by other APs can be used in two ways, simultaneously and concurrently. Actions can be added to another pool so that the original pool is waiting and retaining its Resource for the remote Action. It waits until the Action is executed in the other AP. This way, Actions that use both Resources simultaneously can be executed. Another way is to add Action to the remote pool and just continue execution from the original pool without waiting. The remote Action is considered as one belonging to the task’s set of Actions. This way a task can use several Resources concurrently.

B. World State

For higher level control, the sensor information about the environment has to be abstracted. Abstraction is based on dividing the environment into objects. Objects are stored and updated to a relative database. An important feature of the object representation is its probability information. If the certainty of the size, the existence or the pose of the object is insufficient, it can be observed further until a sufficiently reliable measurement is obtained. This world state database (DB) acts as a virtual shared memory for the distributed components of the system. Instantaneous projections of the world state can be created by inquiring about all objects from the database at one time instance.

C. Event Listener

Event Listeners (EL) are initiated or terminated in the beginning and at the end of each Action. They are similar to “try-catch” statement in programming languages. The difference is that instead of the operating system generating the events, ELs define themselves when an event occurs. EL has two parameters: a threshold value for and a response to the event. Perception agents work in tight co-operation with ELs. Agents analyze the raw sensor information and update the world state accordingly. ELs then follow the world state.

III. EXPERIMENTS

The following sections describes the proof of concept experiments conducted on two different platforms to prove the generality and functionality of the architecture.

A. Experiments with MARY

The first experiment is done with a mobile manipulator robot called MARY [11]. For this experiment, MARY is equipped with a Panasonic pan-tilt-zoom camera unit (PTU) and a Hokuyo URG laser scanner. Player software [12] is used for reactive level control. In a simple experiment, the PTU Resource is divided between two tasks: “to photograph human” and “to search for an object”. The experimental setup is presented in Fig. 3.

In the task “to photograph humans” we detect humans and if one is found, the PTU is directed toward the human and a picture is taken. This picture is then stored as a texture for the human object in the world state. In this task, EL is responsible for watching if a human has entered the scene; in which case it would add a picture taking Action to the AP of the PTU. Meanwhile a human detecting perception agent is continuously updating the world state. Only the reference to the object in the database is communicated between EL and Action.

The perception agent localises and recognizes humans based on the laser scanner. It detects the legs from around knee height and is using several heuristics to differentiate humans from each other and from objects similar to humans. Rules for heuristics include leg size, motion, separation of the legs and distance between two persons. Agent also utilises a Bayesian filter to differentiate humans from objects and to estimate their motion.

In the task “to search for an object”, the PTU is randomly turned to different directions and the object in the camera image is searched for using SIFT feature points [13]. A refined reference set of feature points is acquired beforehand. This task by nature is continuous; meaning that the same Action is added to the AP after its execution. It also has a lower priority than the previously described task of “photographing humans”. So, when the latter task is added to the AP, action selection is triggered resulting in a prompt reaction.

MARY’s pose is also used in a Resource reservation experiment. In this experiment the localisation and navigation is done utilizing particle filter, wave front path planning and vector field histogram of Player package after refining the algorithms for measurement range of the used laser scanner.

Fig. 3. Experimental setup with MARY (left) and Rolloottori (right)

B. Experiments with Rolloottori

Rolloottori is a classic design of differential drive robot originally built for home automation for elderly [14]. Rolloottori is equipped with a Sony PTU and a Hokuyo URG laser scanner. The experiments with Rolloottori involve two APs, Resources of the robot’s pose and PTU’s direction. The GIMnet software [15] is used for reactive level control. Localisation and navigation is based on branch and bound localisation method of [16], road map path planning and reactive multi-stage obstacle avoidance inspired by vector field histogram. The experimental setup is presented in Fig. 3. Below, we explain three of the experiments conducted with Rolloottori in greater detail.

1) Texture mapping a wall segment: The structure of the experimental space with its walls divided into segments are stored in the DB. To have a more accurate model of the
experimental space, these wall segments need to be texture mapped. Considering the distance between the robot and the wall, a wall segment is too big to fit in one frame. Therefore, multiple pictures from different locations are needed to texture a wall segment. Only the DB reference of untextured wall segments are given to the task by the user.

In the first experiment, the robot acquires a texture for a single wall segment. This demonstrates the usage of inter-dependent Resources. The first Action in the task calculates the total number of images required for texture mapping the wall segment and the location and angle from which each image has to be taken. Next, for each of these images a picture taking Action is added to the pose’s AP. The Action corresponding to the image whose location is closest to the robot based on the shortest expected reservation time is selected first (expected execution time for the Plan is the same for all Actions). The robot then reserves the pose Resource by traversing to the picture taking location. At that location, Action’s Plan adds a picture taking Action into the pool of the PTU and waits for its execution. The PTU’s AP then reserves its Resource by directing the camera to the desired section of the wall segment. This process is repeated until all images are taken and then an image stitching Action is added to the pose’s pool. The resulting image is stored in the DB.

The target for Resource to reserve in an Action can be expressed in two ways: i) as a reference in DB or ii) in absolute world coordinates. A reference to the robot itself is used for Actions, such as image stitching, which require only processing power and where the resource is irrelevant.

2) Texture mapping a wall segment with exception: It is inevitable that while Rolloottori is moving autonomously in the laboratory, avoiding objects and taking photographs of walls, the random people in the corridor would also show up in the captured images which would corrupt the final texture mapped wall image. In order to avoid this, a human detecting EL is introduced to be activated during the picture taking Action. This demonstrates exception handling and gradual evolvement of the task in the proposed architecture. Fig. 4 illustrates a part of a "picture taking" task execution while a human approaches the robot. The detailed steps taken in this process are explained below:

Step 1 The first Action calculates locations and angles from which to take images and adds Actions to do just that to the pose AP.

Step 2 Based on the Action selected from the pose AP, the robot is driven to the target pose. The Plan of Action in pose AP adds a "Take Image" Action to the PTU AP. Pose AP awaits for the Action in PTU AP to be executed.

Step 3 Action in PTU AP is selected which initiates "Human too close" EL. A human approaching the robot triggers the EL to add a "Track Human" Action to the PTU AP.

Step 4 Because of the above Action addition, "Take Image" Action is interrupted and "Track Human" Action is started. This Action tracks human with PTU and initiates a "Human too far" EL. This EL is triggered once the human moves further away and as a response, it removes the "Track Human" Action.

Step 5 The "Take Image" Action is reselected and the camera is directed for picture taking. The Plan then saves the image for stitching. After the execution, the Action is removed from both the PTU and pose AP.

Step 6 The pose AP selects a new action for execution. If there are any Actions left in PTU AP, both the pose and PTU AP Actions would be selected and executed in parallel.

3) Texture mapping of wall segments: The last experiment involves multi-tasking using two different tasks: wall segment texture mapping and human greeting. The “human greeting” task is similar to the “photographing humans” task with the difference that one more EL is added. This EL checks if the human also stops by the robot and as a response greets him with an audible greeting. Now there are two functions tracking the human. The behavior of these functions can be tuned by task priorities and ELs’ thresholds. Fig. 5 shows the steps taken by the robot when it “texture maps a wall segment” (which was shown in Fig. 4) and “greets a human” in a multi-tasking fashion. As a result of this additional task, the following would be added on to the previously described steps:

Step 1 The “human greeting” task initiates the “Human too close” EL.

Step 3 The newly added “human greeting” task’s “Human too close” EL is triggered and as a response the “Human too far” and “Human stopped” ELs are triggered.
Step 4a Once the human stops by the robot the "Human stopped" EL adds the "Greet Human" Action.

Step 4b The "Greet Human" Action is selected and executed. When the human moves away, the "Human too far" EL is triggered and it itself initiates the "Human too close" EL in preparation for the next repetition of the above cycle.

With Rolloottori we started with empty wall segments of the experimental space. The robot was moving to different locations for picture taking during the task (Fig. 7 left). As shown on the right hand side of Fig. 7, the stitched texture was easily recognizable. The minor misalignment in the final result was due to the robot’s positioning errors. This could easily be corrected with a more sophisticated stitching algorithm. The current algorithm is using images with information of location, direction and field of view for each captured image and is based on a simple pinhole camera model. The world state after texture mapping some of the wall segments is presented in Fig. 8.

IV. RESULTS

The robots successfully executed all the tasks described in the experiments. Naturally, both robots had a working obstacle avoidance algorithm as part of the pose reservation process. Figs. 6 and 8 present some of the results as a snapshot of the world model saved as an X3D file and viewed with an X3D browser.

Fig. 6 shows the initial state of MARY’s world model which only contained the robot and an empty experimental space. At some time instant, an object was found and added to the DB. As shown in Fig. 6, when a person appeared he was photographed and added to the DB as well. The perception algorithms gave a standard deviation of few centimeters for the object and human locations.

In our experiments, perception proved to be the most challenging part. Firstly, the glass walls in the laboratory often resulted in artifacts that resembled a moving human leg. Secondly, due to the short range of the laser scanner in the long corridors of the laboratory, we got almost identical scans in different locations. Thirdly, the execution time and image quality was affected by the autofocus of the camera. After each camera movement we had to wait for a few seconds for the camera to focus. The autofocus also had some difficulties when photographing walls without any detail. Despite these challenges which are inevitable when using real world equipments, our proposed architecture was able to produce plausible results.

V. DISCUSSION

One might think that shared memory over Ethernet network is too slow for robot control. In our case though, usage of database worked well for information exchange between distributed components. The response delay of the system can be longer when abstraction level is raised. The typical time for one component to write information and for another

![Fig. 5. Content of different action pools and Event listeners during “Texture mapping a wall segments” task: See the text for explanation of the steps.](image)

![Fig. 6. World model with empty experimental space and MARY (left) and projection of MARY’s world model after person and object has been found (right): Texture maps are manually created.](image)

![Fig. 7. Path of the robot for one wall segmenting task (left) and a sample of a stitched wall segment (right): Robot is coming from right and stopping in five locations to take images. The stitch marks in image are due to cameras auto gain but it can still be used as a texture map.](image)
to read was well under 10 ms which is sufficiently fast for our higher level control. A nerve pulse travels from finger tip to brain approximately in the same time. If a faster response time is required, the whole algorithm should be integrated into one component as it is not a higher level control. This is feasible because the sensor information is already easily available via Player or GiMnet.

As with computer programming, the abstraction typically enhances the productivity. We believe that with the proposed architecture robot programming can be abstracted into new level. The advantage of our architecture is that each function can easily be reused, turned on or off, prioritized in wanted order and altered or added without the need to change any other part of the control logic. In conclusion, the proposed architecture is very distributed by nature and thus flexible with its computing requirements.

VI. CONCLUSIONS AND FUTURE WORKS

A successful execution of parallel tasks utilizing interdependent resources was demonstrated using the proposed architecture. Several proof of concept experiments were presented using two different robot platforms with varying underlying architectures. This proved that the proposed architecture can be used for multi-tasking control of service robots.

For future work, we plan to increase the number of perception agents, as a result of which the processing power will also become a critical resource. Computer operating systems already handle the resource allocation gracefully but we aim to construct a mechanism to prioritize certain functions for attention control. We are also planning to implement a nature inspired homeostatic attention control system.

When a task is interrupted it might need a wind down process. For example, when fetching a glass of water the glass has to be put somewhere when the task is interrupted. We are currently investigating how these wind down processes can be added as an EL so that a new urgent task or Action can force the robot to skip the whole wind down process as we humans sometimes do.

REFERENCES