Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 06-027

Enabling Complex Behavior by Simulating Marsupial Actions

Michael Janssen and Nikos Papanikolopoulos

September 25, 2006

Enabling Complex Behavior by Simulating Marsupial Actions

Michael Janssen and Nikos Papanikolopoulos
University of Minnesota Center for Distributed Robotics
{mjanssen,npapas}@cs.umn.edu

2006-09-10

Abstract

Marsupial robotics is a field which requires a high initial investment prior to experiments. We present a new simulation tool used for marsupial actions, enabling research in the area of marsupial robotics. A flexible abstraction for a marsupial system is presented which allows for the modeling of many differing marsupial models. The interface and simulation model are covered in detail. Two experiments are presented which use the model to demonstrate the flexibility and range of the abstraction and simulation.

1 Introduction

Marsupial robotics is an active field of research, allowing for new forms of cooperative robotics. Unfortunately, research in the field is limited by the very nature of marsupial robotics. Performing experiments with a marsupial robotic system is difficult at best. Generally the requirements are at least two heterogeneous robotic platforms and a mechanism for marsupial actions which is often complicated. Many of the desired experiments in marsupial robotics require large numbers of robots, which introduces maintenance and runtime modification. The requirements of a mechanism for marsupial robotics have a wide range of possibilities. Will the mechanism carry one or many robots? What sensors does the mechanism have? Can it charge robots that it is carrying? There are many other questions which are addressed differently by the requirements of each experiment and marsupial mechanism. Many of the marsupial systems in existence today are either simple or incredibly specialized.

The varied requirements and difficulty maintaining large groups of robots desired for marsupial experiments creates an opportunity for robotic simulation software. Utilizing a simulation can ease many of the initial investment and maintenance issues involved in running marsupial experiments. We have developed an extension to the widely-used Player/Stage [11] simulation software which implements a marsupial robotics abstraction, allowing for easier design and execution of experiments.

Section 2 presents earlier work related to simulation and marsupial robotics that we base our work on. Following in Section 3 is an explanation of the design considerations and the final marsupial abstraction which we implemented. Then in Section 4 we describe two experiments using the system: one which deals with resource gathering (Section 4.1) and another which uses the system for energy management (Section 4.2). Finally, we present possible future work in Section 5.

2 Related Work

A number of fields provided the foundation for this work. First, the work in real-world marsupial robotic systems. Additionally, current and continuing works in simulation software. There are also a number of robotic systems developed which may be easier to simulate under the marsupial simulation presented here than with current abstractions in use.

There is a marked lack of published work regarding existing marsupial robotic systems. While some definitions of marsupial systems include reconfigurable robotics such as CONRO [2] or PolyBot [12], our

definition excludes these systems. Most similar to our inspiration for the simulation methodology here are marsupial systems in which the “child” robot is stored inside the “mother” robot. The real-world analogy which is most similar is the Yellow Jackets team [5] from Georgia Tech, in which four AIBOs dock with an ATRV-Mini. Other similar work was performed by Murphy et al. [9] using a retrofitted Power Wheels named Silver Bullet and Bujold, a tethered Inuktun robot stored inside Silver Bullet’s shell until deployed. In addition, systems in which there is a level of autonomy which allows the “child” robots to return to the deployment platform [8] are modeled well by this work.

A number of robots which are not generally classified under the marsupial robotics domain can also be modeled with our software. These include unmanned aerial vehicles which deploy sensor networks [4]. The software also allows for systems which simply collect material such as the Roomba [7] robotic vacuum cleaners.

Arguably the most prevalent simulation software in use today is Player/Stage [11], onto which this work is built as an extension. There are very few robotic simulation systems which include marsupial actions. The easiest example from this class of simulators are full physics simulators such as Gazebo [3]. However, in current simulators with physics, there are still no high-level marsupial abstractions – the entire sequence of manipulators or motors would have to be activated individually in sequence each time a marsupial action was desired. To the authors’ knowledge, there is no other robotic simulation software capable of complex marsupial interaction in existence which does not derive from a full physics simulator.

3 Marsupial Abstraction

Marsupial robotics can be used to describe a wide variety of robotic teams and/or systems. In this case, we would like to present our definition of a marsupial robotic system and define the variety of parameters under which the augmented Stage software can simulate marsupial actions.

We do not consider marsupial systems defined as docking or connecting systems, which also fall under the definition of reconfigurable robotics. By eliminating this set of marsupial systems, we narrow the definition to teams which include a “mother” or dispensing robot and one or more “child” or deployed robotic systems. For the purposes of this paper, we will refer to the marsupial robot and deployed (or stored) objects by these terms. We will also refer to the “bay”, which is a general term for the area in which the objects are stored.

Our abstraction exists in two parts. The Stage model description and simulation has a main focus of making it possible to simulate the widest variety of devices and produce simulated data for output. The Player device interface is used to convey information out of the system and is generally used to specify the device being used.

3.1 Player Device and Interface

The Player server is built out of a number of interfaces which provide interaction to subsystems of a robotic platform. There exist interfaces for most common robot subsystems: The `position2d` interface interacts with 2-dimensional odometry, the `sonar` interface with ultrasonic or infrared sonar devices. However, interfaces may not be abstractions of real-world devices – they can just as easily provide interaction with standard algorithms or other abstractions. One example is the `map` interface, which builds a map using other sensors, another is the `planning` interface, which represents a motion planner in a 2-dimensional space.

The `marsupial` interface is most similar to the `planning` interface in the examples given. It operates at a high level of abstraction, taking commands which may require a large number of actions by other devices in order to execute. The interface is designed to provide information about the state of this high-level abstract device, without care for the actual implementation. This allows the same interface to be used with a wide variety of different real-world methods of marsupial actions.

The `marsupial` interface describes the actions which can be taken by any possible marsupial device. It is intended to be general enough to possibly describe many different systems while being specific to marsupial actions, as opposed to movement or recharging actions. While it may be the case that zero or more actuators are used to perform a marsupial action, the desired action is more high-level.

Table 1: The possible values of the *queue type* field

Value	Meaning
PLAYER_MARSUPIAL_QUEUE_UNKNOWN	Unknown or random placement
PLAYER_MARSUPIAL_QUEUE_FIFO	first stored object is deployed first
PLAYER_MARSUPIAL_QUEUE_LIFO	Stack-like: last stored is deployed first
PLAYER_MARSUPIAL_QUEUE_DEQUE	either the first stored or last stored can be deployed first
PLAYER_MARSUPIAL_QUEUE_PICKPLACE	any object stored can be deployed on command separately

The **marsupial** interface also provides regular data meant to convey the current state of the device, and two request-reply data-sets meant to be queried as needed. The default data which is transmitted from the marsupial interface contains two elements:

1. The state of the bay: an *open* response indicates that the interface is ready to store an object, while a *closed* response means it is not ready.
2. The number of objects stored: this is an integer representing the approximate amount of objects in the bay.

It is only required that the first of these two elements be available. A integer value of -1 is used as a to indicate that there is no way to detect this amount. Note that this number does not need to be an exact value: it could represent the total weight of objects in the bay, or relative fullness in percentage. However, it should generally be used in an upward fashion – using it to indicate space left, or a seemingly unrelated piece of data (the position of an actuator for example) should be avoided.

Other data which is provided by the marsupial interface must be queried for by transmitting a request which then produces a reply. There are two types of request-reply pairs defined at this time. The first is named the *geometry* request and contains:

1. The *outer size* of the bay, representing the footprint on the world that the device takes.
2. The *door size* of the bay, indicating the maximum size object which can be stored.

Either or both of these values may be unapplicable. The second request-reply is named the *storage type* request and contains:

1. The *queue type*, indicating the order in which objects are stored and deployed.
2. The *maximum storage capacity*, indicating at what value the number of objects stored in the standard data state is a full device.
3. The *storage capabilities*, indicating whether a robot can store, deploy or both.

The *queue type* is an enumerated value: the value and meaning of each option are presented in Table 1. Note that the *maximum storage capacity* does not mean that if the number of objects returned by the data state is less, more objects can be stored. A simple example is a double ended queue type in which no more robots can be stored on either end without ejecting a robot from the opposite end.

The *storage capabilities* field is meant to expand the set of devices which may implement the marsupial interface. For example, a vacuuming robot may be modeled as a marsupial interface which allows for the store action, but does not have any sensors in its bay to determine fullness or the amount of dirt stored inside, and of course has no actuators to deploy the objects. In contrast, some marsupial systems [1] can store and deploy a number of robots as well as sense the amount of robots currently stored and possibly communicate with the stored devices. In other marsupial systems [4], the system can deploy robots, but cannot retrieve (store) any robots.

Table 2: descriptors for the marsupial model

keyword	values	description
dropzone_size	$[x, y]$	dropzone size in meters
dropzone_pose	$[x, y, \theta]$	center of the dropzone in meters, and orientation in radians
capacity	x	maximum number of objects stored
stores	0, 1	1 if storage is allowed
deploys	0, 1	1 if deploying is allowed
ordering	<i>fifo, lifo, pickplace, deque</i>	storage and deployment procedure
charges	0, 1	1 if held robots are charged

3.2 Stage model and simulation

The Stage [11] simulator complements the Player robot server by providing a simulation framework in which many (but not all) of the interfaces in Player are available to simulate in a 2-dimensional environment. Each interface which is simulated is connected to a simulation model used by stage to produce data. Models can interact either through Player or directly with the world, allowing for a wide range of simulated devices. Each simulated world is described in a *worldfile*, which is in a tokenizable format parsed by the Stage libraries, and contains the models and environment description used to simulate the entire robotic system. Multiple robots can be simulated in one worldfile, and some or all of the models can be accessed through the Player server.

<pre> marsupial (charges 0 dropzone_size [0.5 0.5] dropzone_pose [0.5 0.5 0] stores 1 deploys 1 ordering lifo) </pre> <p style="text-align: center;">(a) Default marsupial</p>	<pre> marsupial (dropzone_size [1 1] dropzone_pose [0.5 0 0] deploys 0) </pre> <p style="text-align: center;">(b) Vacuum-like marsupial</p>	<pre> marsupial(dropzone_size [3 1] dropzone_pose [0.5 0 0] charges 1 energy()) </pre> <p style="text-align: center;">(c) Charging marsupial</p>
--	---	--

Figure 1: Marsupial worldfile fragments

As part of this work, we have also developed a marsupial simulation model extension to the Stage simulator. This allows for relatively easy experimentation with different marsupial models. The simulated model consists only of a *dropzone* which specifies the area in which objects must be in order to be stored. The model description defines the area where the marsupial interface can store robots, and the location where robots are deployed, as well as defining the possible actions that will be simulated. Figure 1 shows worldfile fragments of two simulated marsupial systems.

The description uses a number of keywords and values in order to define the properties of the simulated system. The properties and their values and effects are shown in Table 2.

The ability of a marsupial system to provide charge to robots stored inside is a particularly difficult situation, because the marsupial interface does not describe an interface for energy transfer. The main reasoning behind this is that there already exists a Player interface for energy status and charging, so we would not like to duplicate these actions, but the simulator can either provide charging capabilities or not. An energy storage method was already in development for Stage at the time of publication and was available in a state as to be usable. Figure 1 shows a portion of a stage world file in which charging has been enabled. The energy available to the robot in general through this simulated energy storage device is used to charge the robot.

4 Experiments

In order to test the performance and flexibility of this new interface and simulation, two experiments were performed using the simulated devices. The first experiment implements a resource gathering task with multiple robots using the marsupial interface to gather the resource objects, meant to simulate the hardware experiments performed in [10]. Next, an experiment in which a larger and marsupial capable robot provided supplies in the form of recharging stations to another group of robots, a simplified version of the marsupial problem addressed in [6].

4.1 Resource Gathering

The stage simulator was setup with a 100 objects which identified themselves as resources. These objects are to be discovered and transported to a common location through the use of a team of 5 robots which are all equipped with marsupial devices. The capacity of the marsupial interface was varied to determine the effect on the efficiency of the task. The start of one run of the experiment is shown in Figure 2.

Figure 2: One start of the resource gathering experiment

The robots run in one of two possible states: *searching* or *returning*. When *searching*, they perform a random walk action unless one or more resources are discovered, in which case they approach the resource and attempt to store it. When they cannot store any more resources, they transition to the *returning* state, in which they use odometry to return to a common drop-off location and deploy the resources.

The total number of resources either stored in a robot or at the home location after 600 seconds is recorded after each run. Preliminary results are shown in Table 3. While it is not shown in the data, it was noticed that there was significantly less contention for the drop-off location when there was less storage space available.

4.2 Energy Management

The simulator is setup with a number of robots with limited amount of energy, which are tasked with the relatively simple task of wall following in a fairly large closed area. Another larger robot is equipped with

Table 3: Resource Gathering Experiment Data

Capacity	Average Retrieved
1	60.3
2	75.3
3	80.4
4	90.5
5	97.6

a marsupial device which can carry one or more portable recharging stations. The recharging stations can charge one robot at a time but are initially placed far from the path taken by wall-following. The larger marsupial robot is tasked with placing the recharging stations so as to minimize the time away from the path. The start of one run of the experiment is shown in Figure 3

Figure 3: One start of the energy management experiment

Each of the smaller robots runs in one of two possible states: *running* or *charging*. When *running*, a simple wall-following algorithm is run. If the robots fall below a specified amount of energy remaining, they transition to *charging* mode. In *charging* mode, the robot moves toward the closest portable charging station and recharges until it has a predetermined amount of energy.

The total amount of time spent in *charging* mode is recorded for each run, as well as the number of energy units dispersed. Preliminary results show that using a marsupial system to place the battery objects drastically reduces the time spent charging.

5 Future Work

Due to the relative early development phase of this software, there are a number of improvements and additions which would be advantageous to develop. One area which is most lacking in the current implementation

is visualization. It should be possible to display the “child” robots which are kept inside the “mother” robot at any point in time. The visualization could simply display the number of stored objects, or be a complicated as displaying each object stored inside in a window of some kind. This area becomes more interesting as you consider the real-world analogies to these visualizations: being able to determine from an outside perspective the objects held inside a marsupial system.

The descriptive power of the world file definition in the marsupial model could easily be extended to include arbitrary polygons as dropzones, or require the robots to be of a certain class. Orientation and size of the objects stored could also be limited in the definition of a marsupial system.

While we specifically excluded reconfigurable robotic systems in our simulation of marsupial systems, it is an ongoing area of research and it may be advantageous to use the marsupial simulation work done here as a starting point for reconfigurable robotic systems. It is possible to create a robot which would perhaps “pick up” another robot and then modify its own size and capabilities to include the capabilities of the now connected system. Other modifications of the properties of the “mother” robot such as an increase in weight leading to more energy consumption for movement should be considered.

It is assumed in our interface and model definition that the “mother” robot in the marsupial system would be in control of storage and deployment. Allowing the “child” robot to initiate or control the marsupial action, as described in more passive docking systems [5, 8], would be a welcome addition.

6 Conclusion

We have presented an extension to the Player/Stage robotic simulator system in order to incorporate marsupial actions. We believe that this work is unique in scope, and useful to stimulate research in marsupial robotics which present unique challenges to real-world research. The extension includes a Player interface definition which eases movement from simulation to real-world experiments, as well as a Stage model definition and implementation which allows for a large range of simulated marsupial interfaces.

7 Acknowledgments

This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. This work has also been supported through the UMN Army Center and the NSF grants #IIS-0219863, #CNS-0224363, #CNS-0324864, and #CNS-0420836.

References

- [1] C. Carlson, A. Drenner, I. Burt, and N. Papanikolopoulos. Modular mobile docking station design. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robot Systems*, 2006.
- [2] A. Castaño, W.-M. Shen, and P. Will. CONRO: Towards deployable robots with inter-robot metamorphic capabilities. *Autonomous Robots*, 8(3):309–324, 2000.
- [3] T. H. Collett, B. A. MacDonald, and B. P. Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian Conference on Robotics and Automation*, 2005.
- [4] P. I. Corke, S. E. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. S. Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. In *IEEE International Conference on Robotics and Automation*, pages 3602–3609, Apr. 2004.
- [5] F. Dellaert, T. Balch, M. Kaess, R. Ravichandran, F. Alegre, M. Berhault, R. McGuire, E. Merrill, L. Moshkina, and D. Walker. The Georgia Tech yellow jackets: A marsupial team for urban search and rescue. In *AAAI Mobile Robot Competition Workshop*, pages 44–49. Edmonton, Alberta, 2002.

- [6] A. Drenner and N. Papanikolopoulos. Docking station relocation for maximizing longevity of distributed robotic teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2436–2441, Orlando, FL, USA, May 2006.
- [7] iRobot corporation. iRobot Roomba® Vacuuming Robot.
- [8] E. Kadioglu and N. Papanikolopoulos. A method for transporting a team of miniature robots. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2297–2302, Las Vegas, NV, Oct. 2003.
- [9] R. R. Murphy, M. Ausmus, M. Bugajska, T. Ellis, T. Johnson, N. Kelley, J. Kiefer, and L. Pollock. Marsupial-like mobile robot societies. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 364–365, Seattle, Washington, 1999. ACM Press.
- [10] P. E. Rybski, A. Larson, H. Veeraraghavan, M. LaPoint, and M. Gini. Communication strategies in multi-robot search and retrieval: Experiences with minDART. In *DARS 2004*, pages 301–310, Toulouse, France, June 2004.
- [11] R. T. Vaughan. Stage: A multiple robot simulator. Technical Report IRIS-00-394, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, 2000.
- [12] M. Yim, D. G. Duff, and K. D. Roufas. Polybot: a modular reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 514–520, Apr. 2000.