A distributed, any-time robot architecture for robust manipulation

Aaron J. Snoswell*, **Vektor Dewanto, Marcus Hoerger, Joshua Song,**
Hanna Kurniawati & Surya P.N. Singh
The Robotics Design Laboratory at The University Of Queensland, Brisbane, Australia
{a.snoswell | vektor.dewanto | m.hoerger | j.song | hannakur | spns} [at] uq.edu.au

Abstract

We present a robot system architecture for a mobile manipulator to perform grasping and manipulation tasks robustly. The system is distributed across multiple computers allowing for online update rates while maintaining a reasonable planning horizon. We test this system architecture using the MOVO mobile manipulation platform from Kinova Robotics. The completed system was demonstrated with a robust manipulation task for over 70 hours at SIMPAR 2018 and ICRA 2018, achieving a success rate ≈ 98%.

1 Introduction

Grasping and placing are key interaction tasks that have received attention from the robotics community. A challenge in this domain is how to handle sensor and actuator uncertainty in a principled and robust manner while maintaining the responsiveness required for real-time, real-world interaction. Here we present the robot software architecture developed as part of our efforts to solve complex robot manipulation tasks under uncertainty. A key enabling factor to solve these tasks is our distributed architecture, which allows us to run an any-time planner off-board the robot, communicating sparse observation and action messages as required.

We demonstrate the merit of this approach with a real-world multi-stage task involving localizing and grasping a cup, scooping some candy from a nearby box, and placing the cup on a table without spillage. Our system achieves robust manipulation by leveraging recent advances in decision-making under uncertainty and by using a state-of-the-art POMDP solver as the underlying motion planner. We realise our system using the MOVO mobile manipulator robot from Kinova Robotics (see Figure 2). The MOVO robot is a mobile manipulator platform consisting of a holonomic wheeled base, a vertical linearly actuated torso, pan-tilt depth sensing 'head' camera, front and rear ground-height linear LIDAR sensors, speakers, a microphone as well as two 6- or 7-DOF Kinova JACO arms. We also present qualitative results obtained with the complete system during two weeks of demonstrations at the SIMPAR and ICRA robotics conferences in 2018.

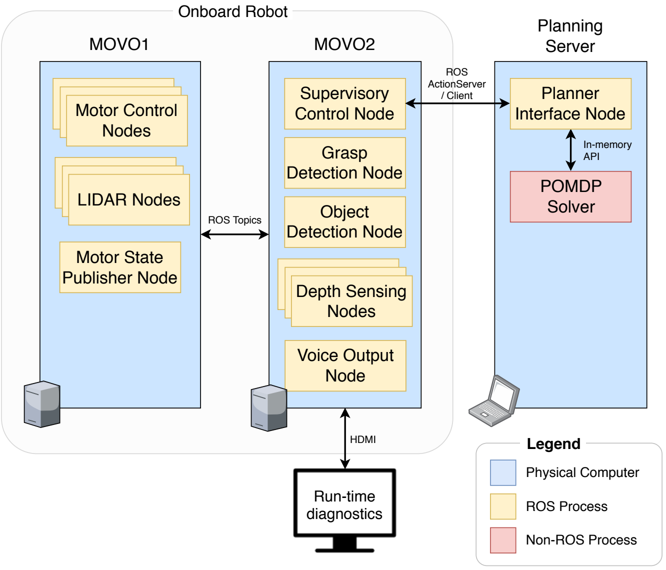


Figure 1: The architecture for our distributed, any-time robot planning system.

The remainder of this paper is structured as follows. In Section 2 we outline the overall nature of our robot system architecture. Section 3 covers how we obtained the sensor measurements required to form observations for the POMDP planner. The coordination of the respective sub-systems with a supervisory control node is outlined in Section 4. Section 5 discusses the interface to the motion planning module. Section 6 discusses our qualitative assessment of the performance of the system, while Section 7 covers related work and Section 8 highlights areas for future work.

*Corresponding author

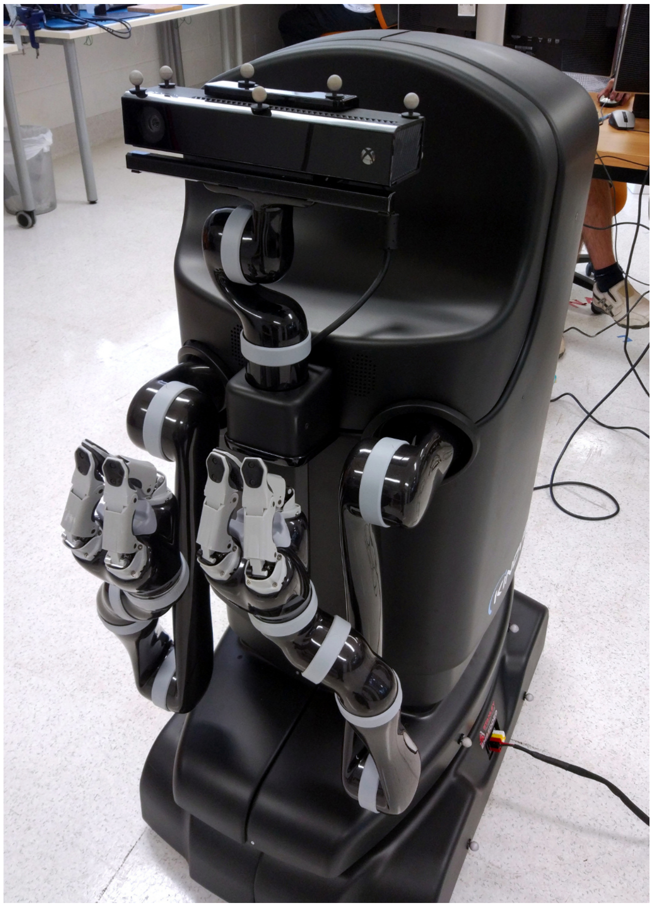


Figure 2: The Kinova Robotics MOVO mobile manipulator robot. Note that the infra-red motion capture marker balls were used only during development and calibration, not in the final system.

2 System Architecture

As stated above, for this project our aim was to demonstrate our POMDP planner using the Kinova MOVO robotic platform. The final architecture we developed to accomplish this is shown in Figure 1. Here, we discuss the different components of this architecture.

2.1 Software and hardware stack

The MOVO robot natively supports the widely-used Robot Operating System (ROS) stack (version 8 'Indigo') [Quigley *et al.*, 2009], so this was selected as the basis for our architecture. The software packages used in our architecture are shown in Table 2, along with version numbers.

The MOVO robot also comes with two on-board Next Unit of Computing (NUC) computers (referred to as MOVO1 and MOVO2), which we leveraged as part of our distributed architecture. As discussed in Subsection 2.2, we opted to include an off-board computer to host the motion planning module. The specifications for the computers used in our final system are outlined in Table 1.

2.2 Load Balancing

Due to a mis-match in supported ROS versions on-board MOVO, we opted to host our planning module on a separate, off-board computer. We decided not to update the MOVO software ecosystem due to time constraints and the fact that the MOVO software was in beta at that point, and thus undergoing rapid changes already.

In the default configuration, the MOVO robot is configured to run motor control and LIDAR processing on MOVO1, while MOVO2 is dedicated to processing incoming point clouds from the RBG+D camera. These processes consumed approximately 15% and 40% of the processor capacity of MOVO1 and MOVO2 respectively. We opted to host the solver on it's own dedicated computer (the planning server), allowing MOVO1 and MOVO2 to remain free to run lower-level robot interface processes. With this configuration, and after adding nodes required for our system, the CPU usage was stable at approximately 20% on MOVO1 and 60% on MOVO2, thus achieving our goal of real-time performance while ensuring the system was safe to use in a public exhibition space.

2.3 Communication with the planner

This distributed nature of the architecture required a communication interface from the planner to the ROS ecosystem on-board the robot. The ROS topic and service features provide for distributed read-only and remote procedure call functionality, however, the long-running and anytime query nature of our POMDP solver meant that these communication methods were not appropriate for our system. Instead, we opted to use the actionlib ROS package, which is designed for long-running pre-emptible tasks[1]. We created a ROS ActionServer node on the planning server and an associated Action-Client on the robot, connected via the standard ROS TCP interface over Ethernet. These nodes then communicated environment observations and requested actions between the planning server and the robot hardware. This required interfacing the stand-alone solver with the ROS ecosystem via a thin wrapper which we refer to as the 'Planner Interface Node' (PIN) (see Section 5).

The observations sent to the planner consisted of a vector $o = (p_o, \theta, \mathit{gripperOpen}, \mathit{targetGrasped})$, where;

- p_o is the 6-DOF pose of the target object to grasp
- θ is a vector of the joint angles for the robot arm
- *gripperOpen* is a boolean indicating if the gripper is currently open or not, and

[1] `http://wiki.ros.org/actionlib`

Table 1: Specifications for the computers used in our system architecture

Computer	MOVO1	MOVO2	Planning Server
Model	Intel NUC5i7RYB	Intel NUC5i7RYB	Dell Precision T3600
Processor(s)	Intel Core i7-5557U @ 3.10GHz (x4)	Intel Core i7-5557U @ 3.10GHz (x4)	Intel Xeon E5-1620 @ 3.60GHz (x8)
Operating System	Ubuntu 14.04.5 64-bit LTS + real time kernel	Ubuntu 14.04.5 64-bit LTS + real time kernel	Ubuntu 16.04.1 64-bit
Disk Space	128GB	128GB	1TB
Memory	16GB DDR3 1867MHz	16GB DDR3 1867MHz	16GB DDR3 1600MHz
ROS Version	8 (Indigo Igloo)	8 (Indigo Igloo)	10 (Kinetic Kame)

- *targetGrasped* is a boolean indicating if the target object is currently being grasped (see Subsection 3.2)

Note that the pose of the gripper is not explicitly passed to the solver, but instead is encoded as part of the joint angle vector θ. Actions received from the planner consisted of vectors $a = (\Delta\theta, gripperAction)$ where;

- $\Delta\theta$ is a vector of absolute angle increments for each joint in the robot arm
- *gripperAction* is an integer indicating the requested gripper action; +1 to fully close, 0 for no action, and -1 to fully open.

The message scripts for ActionServer/Client communication between the planner and the ROS ecosystem are shown in Figure 8.

One benefit of this configuration is the ability to switch out the real robot for a simulation environment. With only minor modifications to the Kinova MOVO ROS configuration, we were able to run our planner for a software-only MOVO robot in the *Gazebo* simulation environment [Koenig and Howard, 2004]. This separation of concerns aided development and debugging of the overall system, reducing the overall development time.

Concretely, we used *Gazebo* v2.2.5. This choice for rather outdated version is because the latest version is not compatible with ROS Indigo. Using an outdated version required manually downloading the default model files as per instructions we found on-line.[2] We simulated the robot's controller using `ros_control` package and a simple *Gazebo* plugin adapter. Note that *Gazebo* requires relatively high specifications for a graphic card.[3] For our application, we found that a dedicated graphic card with 1536 cores and 4 GB of video memory was sufficient.

A challenge we encountered was that the ROS ActionServer/Client message transform structures can loose synchronization if the clocks on the respective computers are misaligned. We found that installing and configuring the clock-synchronisation package `chrony` for Ubuntu fixed this issue.[4] Specifically, we forced MOVO1 and MOVO2 to treat the planning server as a Network Time Protocol master server.

Finally, we also implemented a Supervisory Control Node (SCN) to coordinate observation and action messages, and control the other parts of the system (see Section 4). This process was hosted on MOVO2, along with the nodes for object and grasp sensing (see Section 3).

2.4 Communication with hardware drivers

Communication from the SCN to the lower-level hardware drivers was done through the Kinova MOVO APIs. By default, the MOVO is configured to use the *MoveIt!* motion planner that is part of ROS [Sucan and Chitta, 2011]. We modified the appropriate parts of the Kinova MOVO API to replace *MoveIt!* with commands from our motion planner, via the SCN. The modified logical call hierarchy from planner to actuators is shown in Figure 3, along with the update frequencies and communication methods for each call.

3 Sensing pipeline

3.1 Vision system

The MOVO robot is equipped with front and rear SICK TiM5XX series 2D LIDAR sensors, which are used for navigation and mapping. It is also equipped with a Kinect V2 time-of-flight RGB+D camera mounted on a pan and tilt actuator, which is used for localizing objects for manipulation. ROS provides an interface that projects each pixel in the depth map into a point in 3D space, forming a point cloud.

The candy container was localized through a fiducial marker. The height of the remaining candy was then measured by averaging the height of the relevant points in the point cloud.

In order to localize the cup and the table, the Kinect is panned left and right in a pre-set trajectory. Point Cloud

[2] `http://answers.gazebosim.org/question/6870/what-namespace-does-gazebo-expects/?answer=6959`

[3] `https://answers.ros.org/question/9243/gazebo-supported-graphics-card/`

[4] `https://chrony.tuxfamily.org/`

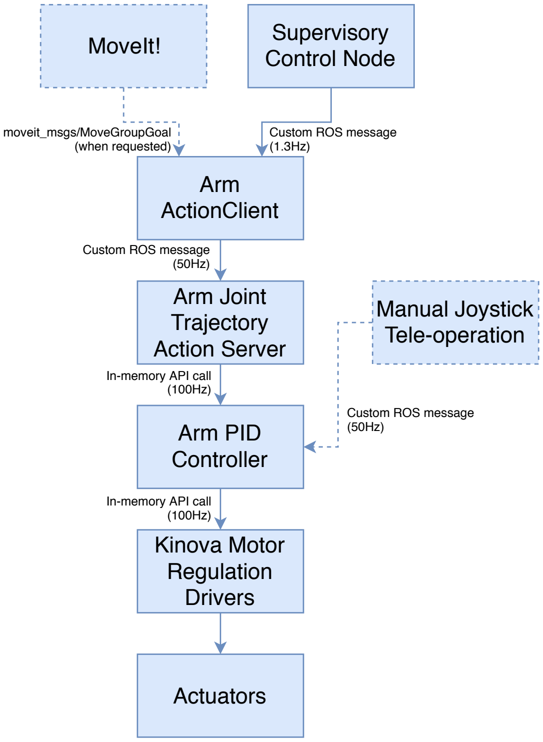

MoveIt!
Supervisory Control Node
moveit_msgs/MoveGroupGoal (when requested)
Custom ROS message (1.3Hz)
Arm ActionClient
Custom ROS message (50Hz)
Arm Joint Trajectory Action Server
Manual Joystick Tele-operation
In-memory API call (100Hz)
Custom ROS message (50Hz)
Arm PID Controller
In-memory API call (100Hz)
Kinova Motor Regulation Drivers
Actuators

Figure 3: The modified call hierarchy to route our motion planner's commands to the MOVO arm actuators.

Library (PCL) v1.7 [Rusu and Cousins, 2011] functions were used for processing the point cloud. This process is shown in Figure 4.

The cup mesh is loaded as a point cloud and down-sampled through a voxel grid filter. Down-sampling reduces the computation required in later steps. The point cloud from the Kinect is filtered to remove statistical outliers and then also down-sampled through a voxel grid filter to the same grid size as the cup point cloud. Self-filtering is performed in order to remove points corresponding to the robot's arm. The table surface points were then segmented using the RANSAC algorithm [Fischler *et al.*, 1981]. A clustering algorithm was then run on the non-table points, and the similarity of each cluster to the cup was calculated through a nearest neighbour algorithm.

Sensor error is a major challenge in manipulation, and the Kinect is no exception. There seems to be conflicting reports from studies quantifying the Kinect's error. [Wasenmüller and Stricker, 2016] measured an almost constant offset of 1.8 cm regardless of depth but [Breur *et al.*, 2014] claims that the error is significantly dependent on the depth. Both papers agree that sunlight and varying sensor temperature can cause error. Sharp object curvature or highly reflective surfaces can also cause erroneous measurements or 'flying pixels'. For example, note the 'fuzziness' around the cup in Figure 10, which contributes to error in estimated object position.

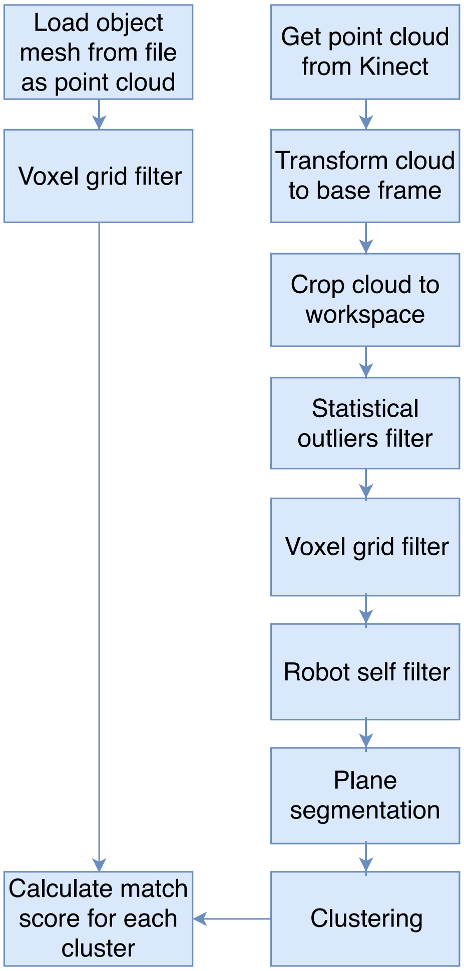

Load object mesh from file as point cloud
Voxel grid filter
Get point cloud from Kinect
Transform cloud to base frame
Crop cloud to workspace
Statistical outliers filter
Voxel grid filter
Robot self filter
Plane segmentation
Clustering
Calculate match score for each cluster

Figure 4: The flowchart for the system vision pipeline.

We conducted an experiment to quantify the error in our object pose estimation. A cylindrical snack can was placed at varying positions on the table and the pose estimate from the Kinect recorded. The ground truth was measured through an OptiTrack motion capture system. The results are shown in Figure 5, where the x-axis is forward from the center of MOVO. The average error is approximately 3cm, but as can be seen, the error depends on the distance and angle of the object relative to the Kinect sensor. The pose estimation error can be increasingly unpredictable if the object is too close to the Kinect. In our case, we did not hard-code any calibrated offsets, and instead handle this significant uncertainty through our POMDP motion planner.

3.2 Grasp detection

The KG-3 end effector provided with the Kinova JACO robot arms is a three-finger gripper, where each finger contains two rotational joints (base and knuckle), but the all three fingers are driven by a single motor (see Fig-

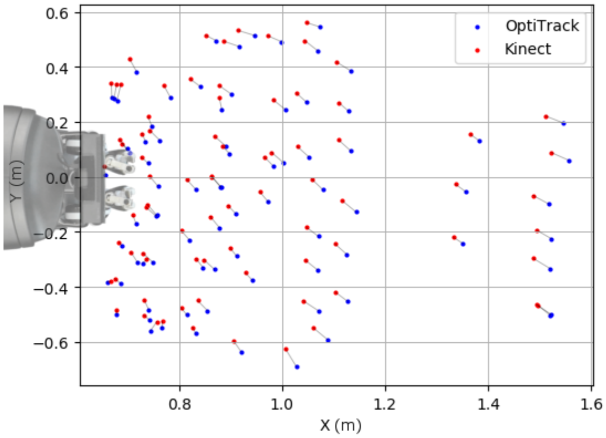


Figure 5: A top-down scatter plot comparing Kinect and OptiTrack pose estimates of an object. The MOVO robot is shown approximately to scale for reference.

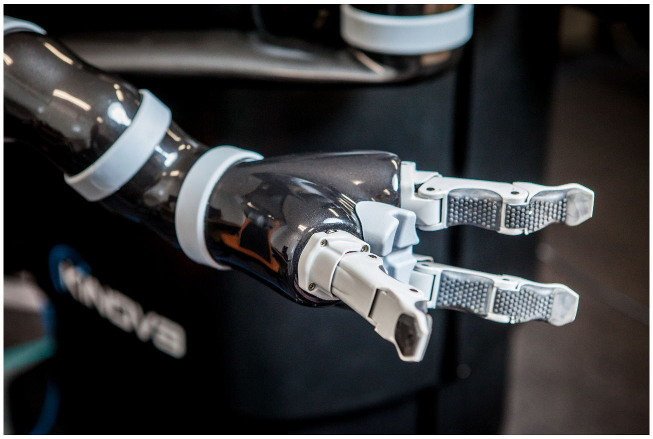


Figure 6: A close-up showing the under-actuated, three-finger KG-3 gripper on the Kinova JACO arm.

ure 6). As such, the gripper is inherently under-actuated. As discussed in Section 4, the Supervisory Control Node was required to send observations to the planner, including whether or not the current target object had been successfully grasped. This presented a challenge, as the default grippers on the MOVO robot do not have any tactile sensing ability, and we did not want to modify the end effectors with external sensors. The end effector interface did provide a reading of the instantaneous current draw for the gripper motor, which given a motor set-point, is theoretically proportional to grasp torque, however we found this signal was too noisy to provide a reliable indication of grasp status. In the final system we used a very simple solution: hard-code the motor position for ‘gripper closed’ and ‘gripper open’ poses, then rely on the compliance in the under-actuated fingers to establish a robust grasp. In addition, we selected as our target grasp object a cup with a small amount of compliance to avoid over-driving the gripper motor.

4 Supervisory control node

The main entry point for the robotic system was a Supervisory Control Node (SCN), which initialized other sub-systems and coordinated high-level control. In particular, the SCN hosted the ROS ActionClient that was used to communicate with the Planner Interface Node (PIN). After initialization, the SCN would continually send updated observations to the planner, and in return, execute requested actions. These processes are illustrated in Figure 9.

5 Planner Interface Node

To ensure a seamless communication between the SCN detailed in section 4 and the planner, we use a Planner Interface Node (PIN). The primary purpose of this node is to serve as a translation unit between the SCN and the underlying planner, converting data structures into ROS messages and vice-versa. Since the PIN is running on the planning server, it also launches the actual planner. As a communication point between the SCN and the planner, it provides two ROS action servers: One that is being used by the SCN to initialize the environment and robot model maintained by the planner after the initial scan has been performed. The second action server is being used by the SCN to provide the planner with the latest observation and, within the same call, requesting the next action. Additionally the PIN informs the SCN about possible planning failures. The PIN is designed to be as thin and lightweight as possible to make sure that all system resources are dedicated to the planner.

5.1 Robust Planning

As outlined above, the robot is subject to various errors in control and sensing, such as imperfect joint encoders and imprecise knowledge of the pose of the target grasp object. In order to enable robust operation in the presence of these uncertainties, we formulate the planning problem as a Partially Observable Markov Decision Process (POMDP). A POMDP is a general and systematic framework for planning under uncertainty that enables the robot to reason about the best action to perform when perfect state information is unavailable. For our problem we use ABT [Kurniawati and Yadav, 2013], implemented within the OPPT framework [Hoerger *et al.*, 2018]. ABT is considered one of the fastest on-line POMDP solvers today and has empirically shown to be significantly more robust against aforementioned uncertainties in our experiments, compared to a deterministic planner, particular for our task where the state and observation spaces are continuous. Additionally it supports our intendend any-time capabilities, meaning that new actions are readily available once the robot has executed the current action.

6 Experimental Results

We qualitatively evaluated the performance of our architecture by running the grasp-scoop-place demo (see Section 1) at the SIMPAR and ICRA robotics conferences in May 2018. Over two weeks we ran this demonstration continuously at the respective conference exhibition halls for approximately nine hours a day. We estimate the system completed a full execution of the demo upwards of 1000 times, achieving a success rate of approximately 98%, the most common failure case being grip slippage. During these demonstrations, we allowed the solver a planning time horizon of 750ms per action. Given that the complete policy for the grasp-scoop-place task required approximately 100 steps from the solver, the total demonstration took approximately 1-2 minutes each time. The memory consumption of the POMDP solver never exceeded 1500MB during the execution of the runs.

Also of interest is the nature of the policies enabled by our architecture. Figure 7 shows an exemplar trajectory for the grasp-scoop-place demo. As can be seen in the figure, we observed that the solver discovered nonholonomic pushing behaviours to increase grasp robustness during the grasping phase. This is a result of the uncertainty encoded in the observation and actuation models, and not a result of manual reward or policy engineering, demonstrating one of the benefits of using a planer that accounts for model and observation uncertainty.

7 Related Work

Robotic manipulation is a research area with a rich history. A good overview of early, decision-theoretic approaches is given by [LaValle, 2006]. More recently, there is been a trend to integrate task and motion planning in manipulation domains [LaValle and Hutchinson, 1998]. For example, [Kaelbling and Lozano-Pérez, 2013] describe an approach based on symbolic operators over belief spaces.

POMDP models for motion planning have historically been applied to problems with limited degrees of freedom (e.g. navigation). For example, see [Thrun *et al.*, 2005] for a good overview, or [Kurniawati *et al.*, 2011] for an extension to longer-horizon and larger state-space problems. The relatively recent emergence of approximate (sample-based) and efficient solvers has enabled application to higher degree-of-freedom problems, such as those encountered in manipulation tasks, as demonstrated in our case. However other examples of this application are sparse in the literature. [Pajarinen and Kyrki, 2017] also use a POMDP as the motion planning module for a vision-based real-world robotic manipulation task with a Kinova robotic arm. The primary difference with our work is that they utilize a custom online POMDP solver, while we use the ABT solver by [Kurniawati and Yadav, 2013] implemented within the OPPT framework [Hoerger *et al.*, 2018]. They also focus on a multi-object manipulation task, while our focus is on a single-object task with multiple types of manipulation (grasping, scooping and placing). [Monsó *et al.*, 2012] also utilise a POMDP for manipulation planning, however their problem domain is deformable clothing, while we focus on manipulation of rigid bodies (a cup) in the presence of deformable clutter (a container full of small candy).

In contrast, [Koval *et al.*, 2016], simplify the manipulation problem by constraining the end effector to a fixed transformation relative to the support surface and build a lattice in configuration space. Their approach relies on real-time feedback from contact sensors and they solve the POMDP using the DESPOT planner [Ye *et al.*, 2017] by leveraging two key ideas; (a) lazily constructing a discrete lattice in the robot's configuration space, and (b) guiding the search with heuristics.

8 Conclusion

We have presented our architecture enabling robust manipulation for a Kinova MOVO manipulator platform. This distributed architecture enabled us to robustly accomplish a multi-stage manipulation task on a real-world platform with significant sensor error.

Here we summarise some of the lessons learned during our development process;

- **Reference frame for motion capture**: In order to obtain ground-truth poses of the robot and its environment during calibration and development, we used the OptiTrack tracking software. This requires a single marker attached to the robot chassis, which is used as reference for the robot frame. Ideally, the robot URDF should contain a frame at which to place the reference marker, and the physical location of this point should be clearly marked, easy to reach and visible from many camera angles.
- **Wireless emergency button**: For a mobile manipulator, it is not sufficient to have a single onboard emergency button; this requires a user to be on standby *near* the robot in the event of an emergency. We suggest that with such systems, a wireless emergency button is crucial to allow shutting down the robot from some reasonable distance. Note that although wireless, this emergency mechanism should maintain a direct access to the internal power circuits, bypassing any application layers. This implies, for example, a conventional joystick control loop can not be used.
- **Ergonomic charging point**: Power charging is arguably one of the most routine robotics task per-

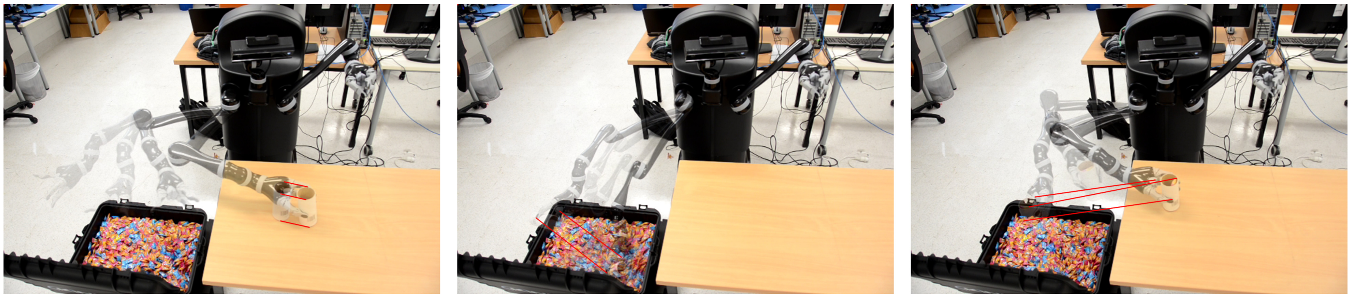


Figure 7: Images showing the robot executing the grasp (left), scoop (middle) and place (right) actions. Red lines track physical features for clarity. In the grasp image, the learned non-prehensile pushing behaviour can be seen at the end of the trajectory.

formed by human operators. Therefore, we believe that having an ergonomic charging point is essential. Its design should also take into account possible extensions towards autonomous charging.

- **Migration to the latest ROS Long-Term-Support (LTS) version**: Our MOVO robot came with ROS Indigo ecosystem, which is already outdated by one newer LTS version. In retrospect, we believe there may have been significant time-savings to perform a system migration to a newer LTS version before commencing development. We encountered significant technical problems integrating our software with the outdated ROS packages. An obsolete ROS ecosystem also leads to 'domino effects'. For example, it requires an old Ubuntu version that brings old packages, including the compilers, `git`, `cmake` etc. Additionally, third-party ROS packages with relevant updated features are usually tied to specific newer versions and not backward compatible.
- **Support for wide range of physics engines and simulators**: Simulation is an integral part in the robot development process. In recent years, there has been a significant progress in the development of physically accurate physics engines and simulators. For instance, in addition to robot simulators like *Gazebo*, the community also enjoys powerful physics engines with advanced features such as Mujoco and PyBullet[5]. Unfortunately, those varieties of simulators do not use standardized robot model formats. This means switching from one to another is not a trivial task. We therefore suggest robot manufacturers should aim to support a wide-range of physics-engines and simulators, and developers of physics engines should strive to standardise support for common robotic model specification files.

We hope these suggestions can guide future developers working with mobile manipulator platforms.

Considering possible areas for future work, there are a number of interesting avenues. In our demo, we only used the POMDP solver for planning joint trajectories for one arm of the MOVO robot. Time constraints prevented us from fully exploiting the degrees of freedom on the MOVO mobile manipulator platform. It would be worthwhile to study the types of behaviours that could be learned if both arms could be used (e.g. for bi-manual manipulation tasks), as well as the base and torso actuation. Likewise, the flexibility of the POMDP model can allow agents to exhibit information-gathering behaviours [Spaan, 2008]. Moving our visual scan step (see Figure 9) to be part of the inner planning loop observations would potentially allow for active sensing behaviours.

Another area of much interest is the use of modern convolutional deep-learning approaches for robotic vision tasks. Replacing our vision pipeline with a convolutional deep neural network model would present interesting research challenges in the form of different structure in the observation errors.

While a small initial step, we hope this result adds to the growing body of evidence that robust manipulation and planning is feasible, even under real-world uncertainty and with high-dimensional state and action spaces.

Acknowledgements

Our thanks to Kinova Robotics for technical assistance with this project as part of the MOVO beta program. M. Hoerger is supported by a joint Research Training scholarship from The University of Queensland and the Australian Government Commonwealth Scientific and Industrial Research Organisation. V. Dewanto is supported by a University of Queensland Research Training Scholarship. J. Song is supported by an Australian Government Research Training Program Scholarship. A. Snoswell is supported by an Australian Government Research Training Program Scholarship and a University of Queensland School of ITEE Scholarship.

[5] `http://www.mujoco.org` and `https://github.com/bulletphysics/bullet3`

References

[Quigley *et al.*, 2009] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. *ROS: an open-source Robot Operating System.* In ICRA workshop on open source software, vol. 3, no. 3.2, p. 5. 2009.

[Koenig and Howard, 2004] Nathan P. Koenig, and Andrew Howard. *Design and use paradigms for Gazebo, an open-source multi-robot simulator.* In IROS, vol. 4, pp. 2149-2154. 2004.

[Sucan and Chitta, 2011] Ioan A. Sucan and Sachin Chitta *MoveIt!.* [Online] Available at: http://moveit.ros.org, [Accessed 28 Sep. 2018].

[Rusu and Cousins, 2011] Radu Bogdan Rusu and Steve Cousins *3D is here: Point Cloud Library (PCL).* In IEEE International Conference on Robotics and Automation (ICRA), 2011.

[Fischler *et al.*, 1981] Martin A. Fischler, and Robert C. Bolles *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography.* Communications of the ACM 24, no. 6 (1981): 381-395.

[Wasenmüller and Stricker, 2016] Oliver Wasenmüller and Didier Stricker. *Comparison of kinect v1 and v2 depth images in terms of accuracy and precision.* In Asian Conference on Computer Vision, pp. 34-45. Springer, Cham, 2016.

[Breur *et al.*, 2014] Timo Breur, Christoph Bodensteiner, and Michael Arens. *Low-cost commodity depth sensor comparison and accuracy analysis.* In Electro-Optical Remote Sensing, Photonic Technologies, and Applications VIII; and Military Applications in Hyperspectral Imaging and High Spatial Resolution Sensing II, vol. 9250, p. 92500G. International Society for Optics and Photonics, 2014.

[Kurniawati and Yadav, 2013] Hanna Kurniawati and Vinay Yadav *An online POMDP solver for uncertainty planning in dynamic environment.* Proc. Int. Symp. on Robotics Research 2013.

[Hoerger *et al.*, 2018] Marcus Hoerger, Hanna Kurniawati, Alberto Elfes *A Software Framework for Planning under Partial Observability.* In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[LaValle, 2006] Steven M. Lavalle *Planning Algorithms.* Cambridge university press, 2006.

[LaValle and Hutchinson, 1998] Steven M. Lavalle and Seth A. Hutchinson *An objective-based framework for motion planning under sensing and control uncertainties.* The International Journal of Robotics Research 17, no. 1 (1998): 19-42.

[Kaelbling and Lozano-Pérez, 2013] Leslie P. Kaelbling and Tomás Lozano-Pérez *Integrated task and motion planning in belief space.* The International Journal of Robotics Research 32, no. 9-10 (2013): 1194-1227.

[Koval *et al.*, 2016] Michael Koval, David Hsu, Nancy Pollard and Siddhartha Srinivasa *Configuration Lattices for Planar Contact Manipulation Under Uncertainty.* In Workshop on the Algorithmic Foundations of Robotics, 2016.

[Ye *et al.*, 2017] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee *DESPOT: Online POMDP planning with regularization.* Journal of Artificial Intelligence Research 58 (2017): 231-266.

[Thrun *et al.*, 2005] Sebastian Thrun, Wolfram Burgard, and Dieter Fox *Probabilistic robotics.* MIT press, 2005.

[Kurniawati *et al.*, 2011] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee *Motion planning under uncertainty for robotic tasks with long time horizons.* The International Journal of Robotics Research 30, no. 3 (2011): 308-323.

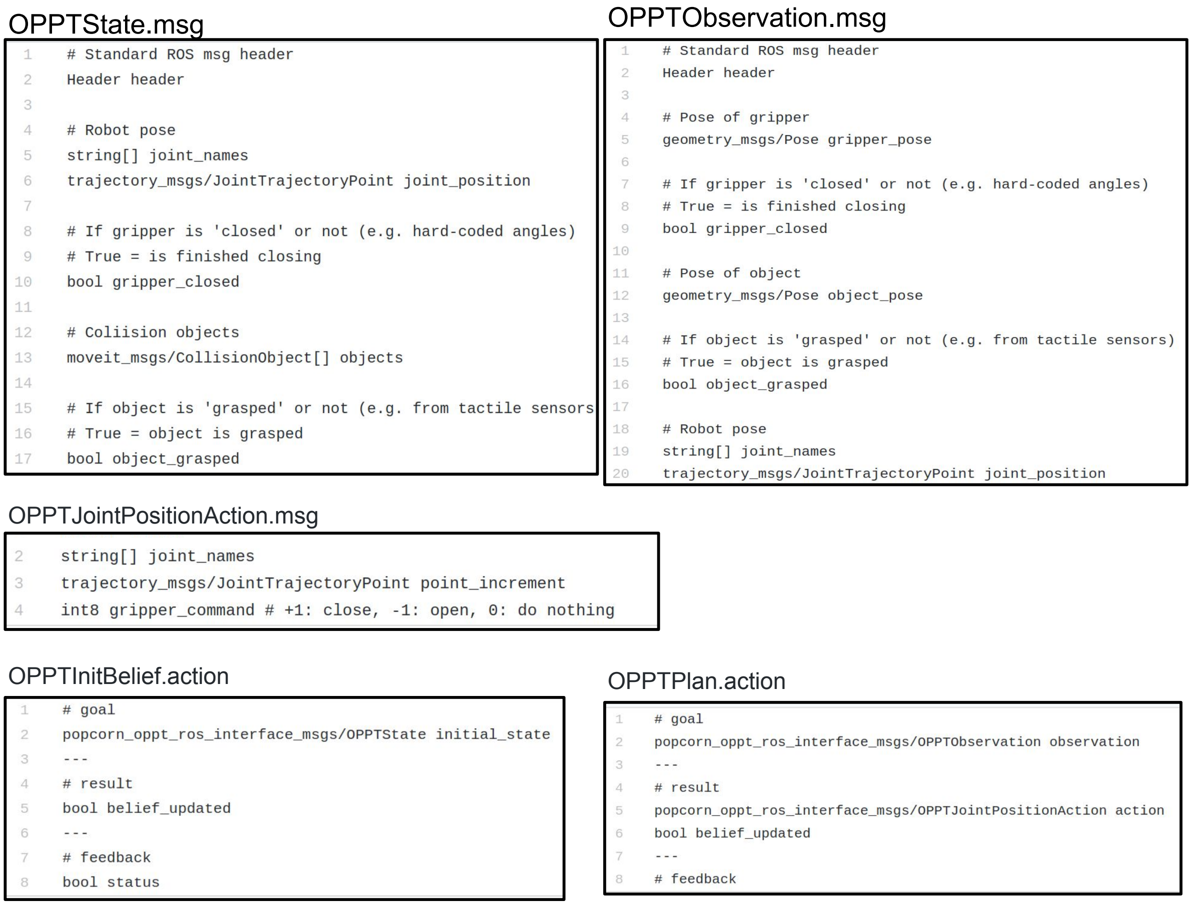
[Pajarinen and Kyrki, 2017] Joni Pajarinen and Ville Kyrki *Robotic manipulation of multiple objects as a POMDP.* Artificial Intelligence 247 (2017): 213-228.

[Monsó *et al.*, 2012] Pol Monsó, Guillem Alenyá, and Carme Torras *POMDP approach to robotized clothes separation.* In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 1324-1329. IEEE, 2012.

[Spaan, 2008] Matthijs TJ Spaan *Cooperative active perception using POMDPs.* In AAAI 2008 workshop on advancements in POMDP solvers. 2008.

Table 2: The primary ROS packages used in our architecture

Package (Indigo branch)	Description
`ros_control`	Controller interfaces, controller managers, transmissions and hardware interfaces
`controller_manager`	A hard-realtime-compatible loop to control robot mechanisms
`gazebo_ros`	Message and service publishers for interfacing with *Gazebo*
`pcl_ros`	Bridge for 3D applications involving n-D Point Clouds and 3D geometry processing
`simple_grasping`	Basic capabilities and boilerplate codes for grasping
`head_action`	An action interface for pointing the head of the configured robot
`moveit_kinematics`	Inverse kinematics solvers in *MoveIt!*
`tf`	Tracker for multiple coordinate frames over time
`actionlib`	A standardized interface for interfacing with pre-emptible tasks
`joint_state_publisher`	For setting and publishing joint state values for a given URDF
`robot_state_publisher`	For publishing the state of a robot to `tf`
`rviz`	3D visualization tool
`joy`	Driver for a generic Linux joystick
`geometry_msgs`	Messages for common geometric primitives such as points, vectors, and poses
`trajectory_msgs`	Messages for defining robot trajectories



OPPTState.msg

```
# Standard ROS msg header
Header header

# Robot pose
string[] joint_names
trajectory_msgs/JointTrajectoryPoint joint_position

# If gripper is 'closed' or not (e.g. hard-coded angles)
# True = is finished closing
bool gripper_closed

# Coliision objects
moveit_msgs/CollisionObject[] objects

# If object is 'grasped' or not (e.g. from tactile sensors
# True = object is grasped
bool object_grasped
```

OPPTObservation.msg

```
# Standard ROS msg header
Header header

# Pose of gripper
geometry_msgs/Pose gripper_pose

# If gripper is 'closed' or not (e.g. hard-coded angles)
# True = is finished closing
bool gripper_closed

# Pose of object
geometry_msgs/Pose object_pose

# If object is 'grasped' or not (e.g. from tactile sensors)
# True = object is grasped
bool object_grasped

# Robot pose
string[] joint_names
trajectory_msgs/JointTrajectoryPoint joint_position
```

OPPTJointPositionAction.msg

```
string[] joint_names
trajectory_msgs/JointTrajectoryPoint point_increment
int8 gripper_command # +1: close, -1: open, 0: do nothing
```

OPPTInitBelief.action

```
# goal
popcorn_oppt_ros_interface_msgs/OPPTState initial_state
---
# result
bool belief_updated
---
# feedback
bool status
```

OPPTPlan.action

```
# goal
popcorn_oppt_ros_interface_msgs/OPPTObservation observation
---
# result
popcorn_oppt_ros_interface_msgs/OPPTJointPositionAction action
bool belief_updated
---
# feedback
```

Figure 8: The message scripts for action server/client communication between the planner and the ROS ecosystem.

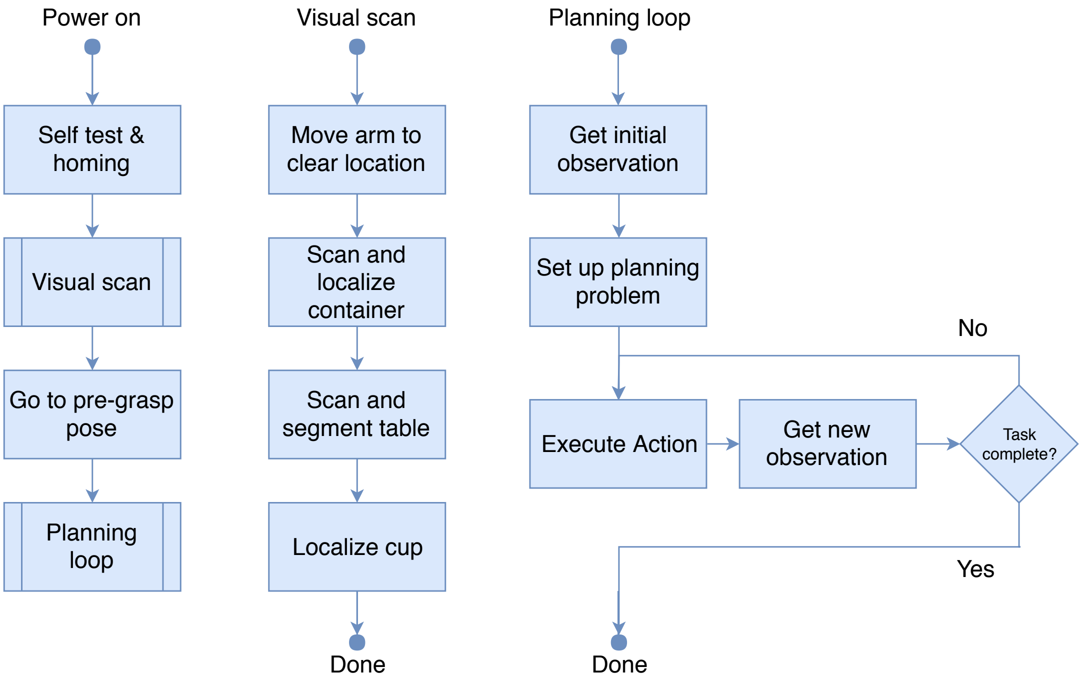


Figure 9: Process flow charts for power-on (left), initial visual scan (middle), and the planning loop (right).

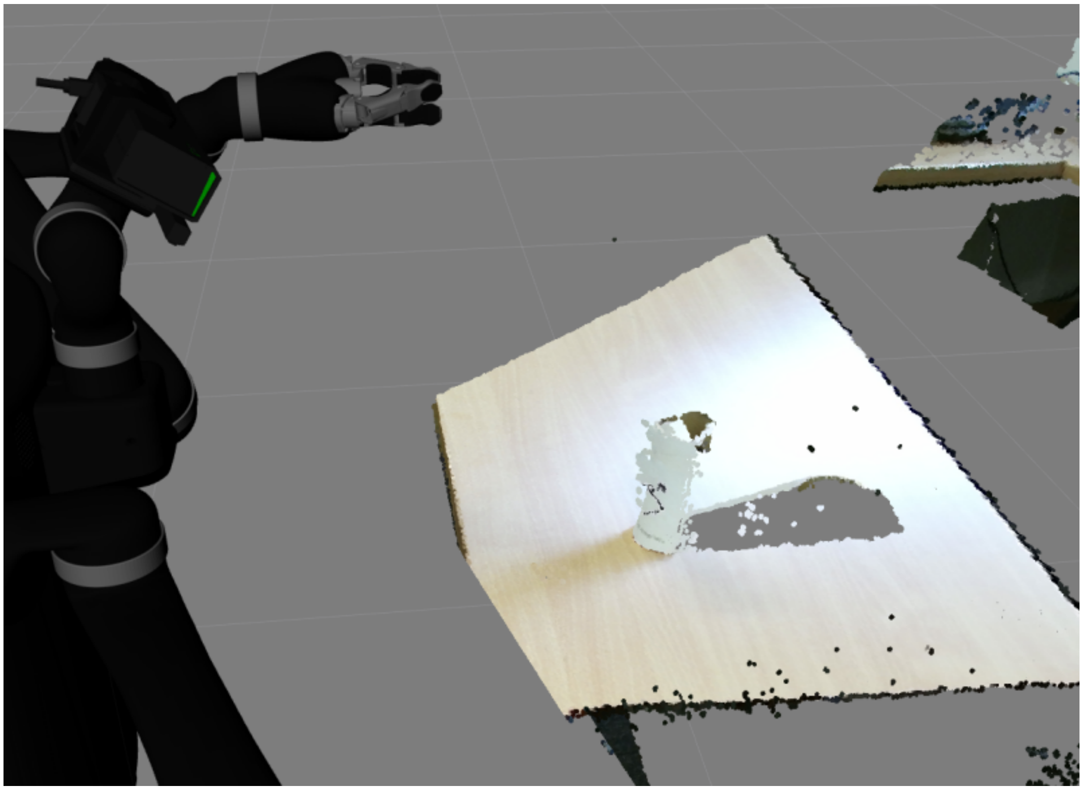


Figure 10: A screen shot showing the noise in the point clouds returned from the Kinect RGB+D sensor. By accounting for this sensor uncertainty in the POMDP model, our system was able to robustly grasp, even varying lighting conditions.