ARC@Work

RoboCup@Work 2017 Team Description Paper

Farzad Ahmadinejad, Mir Alireza Athari, Alireza Zamanian, Amirreza Kebritchi, Saeed Bakhshi Germi, Farzam Janati, Ali M. Hosseini, Sara Havashi Nejadian and Saeed Shiry Ghidary

Robotics Research Center, Amirkabir University of Technology, Hafez Street, Tehran, Iran.

Email: f.ahmadinejad@aut.ac.ir

Web: http://arc-tech.ir/

Abstract. This paper presents the ARC@Work team. We describe the current state of robot hardware and software as well as plans and goals toward 2017 RoboCup@Work competition. The hardware is based on Sepanta, which has been used as a Service robot in @home league previously, but it's altered to suit the @work tasks. The software architecture is based on ROS framework. The research of team is focused on improving the current algorithms to increase robustness.

1. Introduction

The ARC team was founded in 2011 at the Robotic Center of Amirkabir University of Technology, by graduate students in fields of electronics, computer science, and mechatronics engineering. The focus of the team has been MAVs and @home robot so far, but we decided to concentrate on robots with industrial usage. Our short time goal before RoboCup 2017 competitions is to attend RoboCup IranOpen 2017 competition in @work league.

Over the years we participated in different national and international events. Here is a list of most recent and prestigious awards:

- 2nd place award at IMAV 2015 indoor section under the name "ARC"
- 1st place award at IranOpen 2014 UAV section under the name "ARC"
- 2nd place award at IranOpen 2014 @home section under the name "AUT@Home"
- Participation at RoboCup 2014 under the name "AUT@Home" (8th place)
- 2nd place award at IranOpen 2013 UAV section under the name "ARC"

Our main research interests include collision avoidance in complex dynamic environments, optimal manipulation in industrial settings and vision based autonomous navigation of robots.

An industrial robot interacts with people, complex objects, and dynamic environment. So, we are trying to improve our perception, manipulation, and fault detection algorithms to improve our robot's functionality.

2. Hardware

2.1. Base platform

Since our robot is based on Sepanta III [1] and is modified, we tend to describe only the base platform of Sepanta III (see Figure 1). To make navigation simpler for the robot, the base platform is designed and implemented as a "Holonomic" system that has four mecanum wheels [2]. To control the base of robot we use an Arduino DUE[1] board that is connected to ROS[2] via a serial port. This board also performs the task of battery management, odometry estimation, and emergency stop.

The movement speed of the base is limited to a maximum linear velocity of 30 cm/s and a maximum angular velocity of 0.68 rad/s. These are not the absolute maximum rates available for the robot, and it can be changed via a parameter if needed. In our tests, these values were fairly sufficient.

All the high-level control and vision software are running on a laptop. It lets the robot to move easily without any off-board process requirement, but to increase the safety of operation, another laptop can be connected to the main one via SSH.

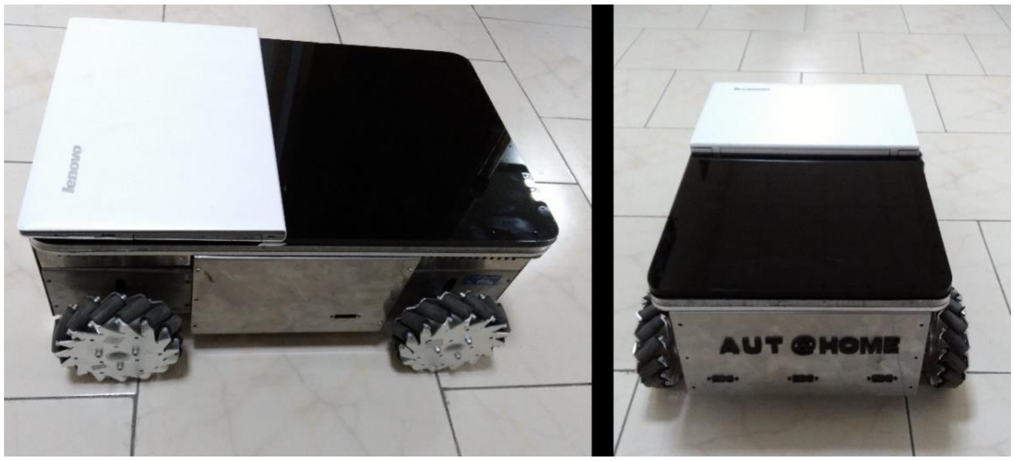


Figure 1. Sepanta III base platform.

[1] To read more about Arduino Due please refer to this link: https://www.arduino.cc/en/Main/arduinoBoardDue
[2] To read more about ROS please refer to this link: http://www.ros.org/

2.2. Manipulator and Gripper

The manipulator intended for Sepanta III lacked the ability to grasp objects directly from above. So, we designed a new arm based on open source platform of uArm[1] with some modifications: We used acrylic instead of metal to decrease the time effort required to build it, and replaced motors with dynamixel[2] to increase torque and accuracy of the arm (see Figure 2). Lastly, we redesigned the end effector so it has two more DOFs (See Figure 3). The designed arm is not failure tolerant itself, but this can be accomplished with software.

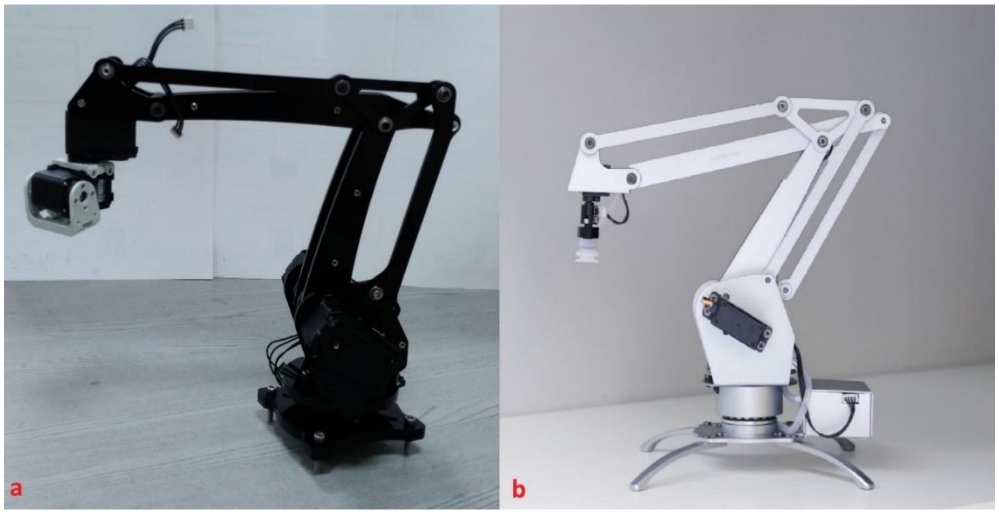


Figure 2. Our manipulator (a) and uArm metal manipulator (b).

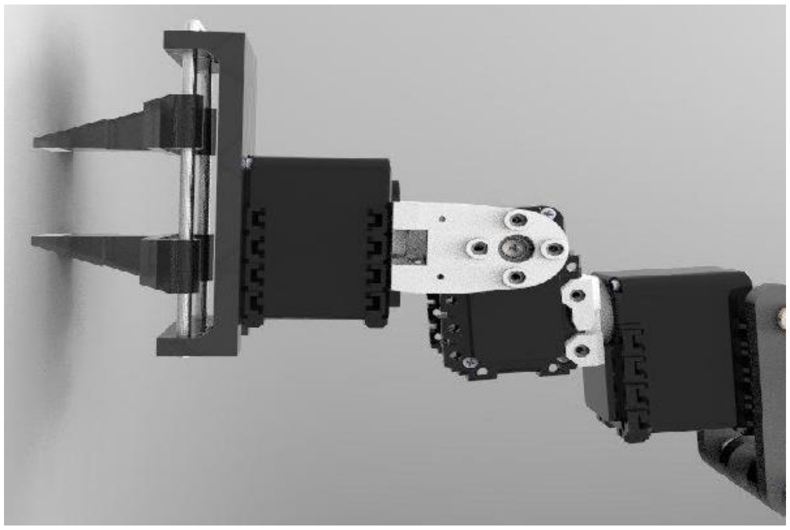


Figure 3. Model of our end effector and gripper in Solidworks.

[1] To read more about uArm please refer to this link: http://ufactory.cc/#!/en
[2] To read more about dynamixel motors please refer to this link: http://www.robotis.us/dynamixel/

In order to grip the object reliably, two steps are integrated on the arm. After the recognition of object and estimation of position and orientation of it, the initial pose information is calculated for the arm and passed to the controller so that the gripper is located above the object. Then, the controller switches to visual servoing mode, in which, an object tracking algorithm keeps the gripper on the object and feeds the controller the essential data (size and weight of object) to grip the object. To verify the gripping, we use the torque feedback of dynamixel motor.
Figure 4 shows the model of completed robot in Solidworks.

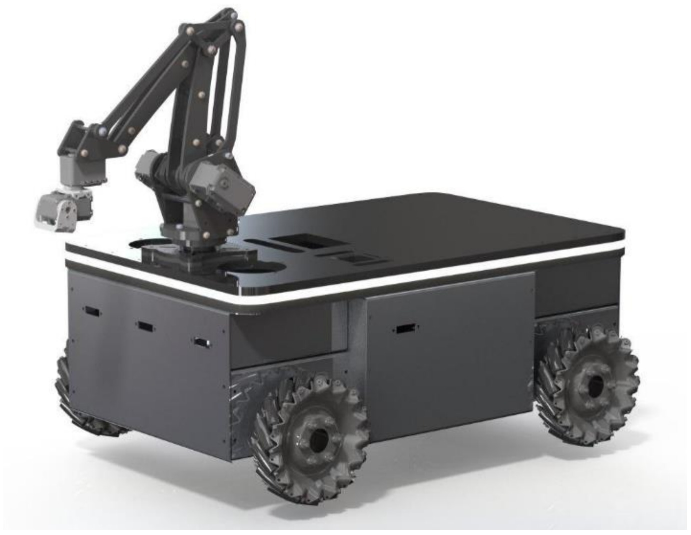


Figure 4. Model of completed robot in Solidworks.

2.3. Sensors

Our robot is equipped with both laser scanner and RGB-D camera, which is used in navigation, perception and recognition tasks. We use the Hokuyo UTM-30LX[1] laser scanner (see figure 5) to implement SLAM. The table below holds the specifications for this laser scanner:

Table 1. Hokuyo UTM-30LX laser scanner specifications

Power Source	12VDC
Detection range	0.1 to 30 m, max 60m, 270^o
Accuracy	30 mm on 0.1 to 10 m, 50mm on 10 to 30 m
Angular Resolution	0.25 degrees
Scan Time	25 msec/scan

[1] To read more about Hokuyo UTM-30LX laser scanner please refer to this link: http://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html

As noted in the table, the detection range is 270^o, and since we don't have backward movement, the robot can move safely using even one laser scanner.

Figure 5. Hokuyo UTM-30LX laser scanner

We also need a camera to identify objects and markers. The Real-Sense camera has been used by many researchers due to its performance. We mounted an R200[1] (see figure 6) on the manipulator. The table below has the specifications for this RGB-D camera:

Table 2. Real-Sense R200 RGB-D camera specifications

	Color	Depth
Active Pixels	1920 x 1080	640 x 480
Frame rate	30 / 60 fps	30 / 60 fps
Field of view (D x V x H)	77^o x 43^o x 70^o (Cone)	70^o x 46^o x 59^o (Cone)

Figure 6. Intel Real-Sense R200 camera

3. Software

The software framework is based on ROS (Robot Operating System) that runs on Ubuntu Linux. ROS has a core that manages inter-process communications among ROS nodes. This will allow us to work on separate, standalone packages for different purposes and then, interconnect them via ROS. We use SMACH[2] state machine system to maintain a task-level behavior of the robot. The currently deployed ROS distribution is Indigo.

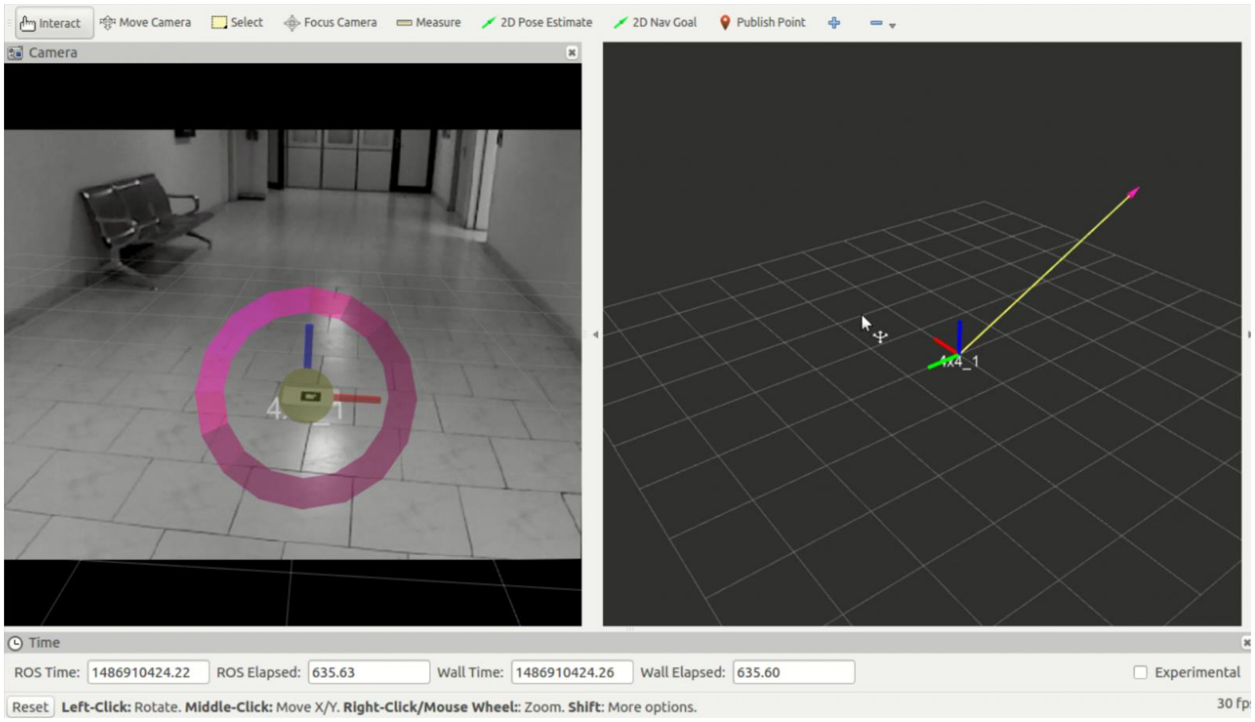
---

[1] To read more about Intel Real-Sense R200 camera please refer to this link: http://reconstructme.net/qa_faqs/intel-realsense-r200-review/

[2] SMACH is a task-level architecture for rapidly creating complex robot behavior. To read more about it please refer to this link: http://wiki.ros.org/smach

3.1. Navigation

The navigation is based on ROS navigation stack[1]. First, we use gmapping[2] to create a map with the laser scanner and odometry data. The AMCL on navigation stack is usually used to get the location of the robot in a map, but we didn't use it, instead Hector-SLAM [3] was utilized to localize the robot. The location of robot is then verified with AR marker detection algorithms. The ARToolKit[3] was used to achieve this goal (see Figure 7).



ROS Time: 1486910424.22 ROS Elapsed: 635.63 Wall Time: 1486910424.26 Wall Elapsed: 635.60

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

Figure 7. ARToolKit Output.

To navigate through the map, we use Move-Base that works based on two cost maps for local and global planner. To choose the path we use the A^* algorithm. After the trajectory is generated, we define mid-goals on it. The base platform tries to reach the final goal by passing through each mid-goal in the defined order. The use of omnidirectional movement enables the robot to move to these goals easily in three steps: 1. rotate toward the path, 2. move on the path, and 3. rotate toward goal position (see Figure 8).

[1] Navigation stack is a package that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base. To read more about it please refer to this link: http://wiki.ros.org/navigation

[2] Gmapping package provides a laser-based SLAM on ROS. To read more about it please refer to this link: http://wiki.ros.org/gmapping

[3] ARToolKit is a library for building augmented reality applications. To read more about it please refer to this link: http://www.hitl.washington.edu/artoolkit/

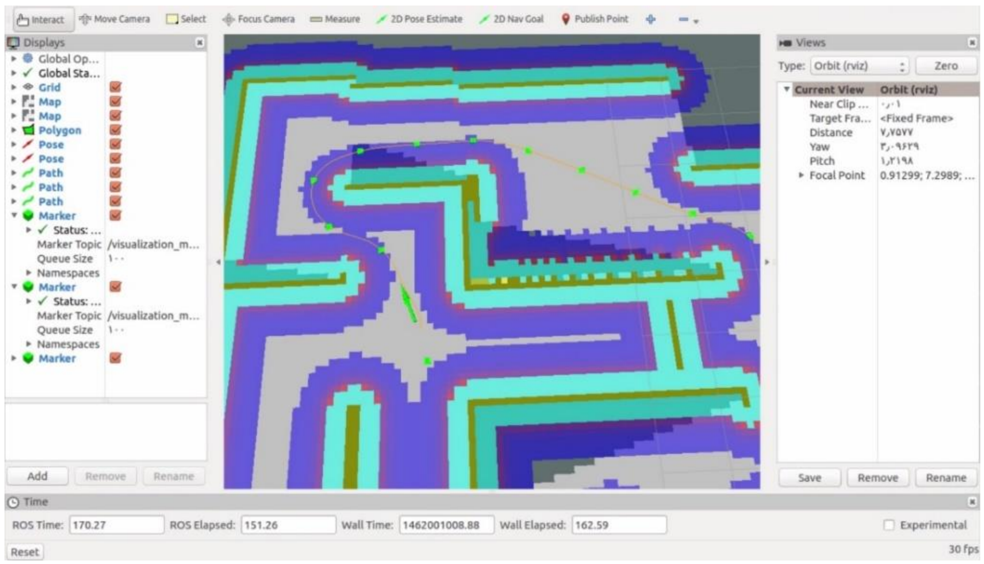


Figure 8. Simulation of our navigation software. The green squares are mid-goals.

3.2. Object perception and recognition

We use the conventional OpenCV[1] library to detect and recognize the objects. First, we position the camera on top of table so it gets the top-down look to objects. Then we convert the image to gray-scale. Finally, we use adaptive template matching [4] to classify them. Figure 9 demonstrates this routine.

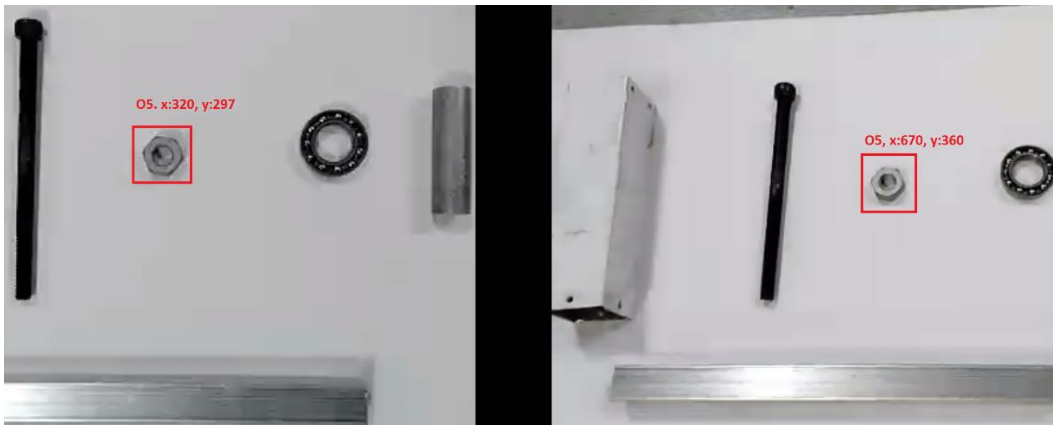


Figure 9. Object recognition.

[1] OpenCV is an open source computer vision and machine learning library. To read more about it please refer to this link: http://opencv.org/

3.3. State machine

The variety of categories in tasks gives a hint about the hierarchical approach that we need to complete these tasks.

After the power supply is connected the robot starts initialization process. When it's finished, the software waits for a command from either referee box or emergency stop. If the command is an emergency stop then robot halts and waits for a reset. Otherwise, the command becomes the next task. Every task is a combination of navigation and manipulation. So a unit decides which subtask to do at the moment.

For navigation subtask, we need two distinct levels. On the higher level, a controller receives the set point for the final position of the base platform and passes it to the lower level. At the lower level, a controller is implemented that navigates the base to reach the final goal. If the task was successful the lower level will send a message to the higher level, and likewise, the higher level sends a message for decision unit. If by any means the task fails, the higher level chooses to repeat it or to abandon it.

The manipulation subtask is somehow similar to navigation. We need two distinct levels. On the higher level, a controller receives the part information and tries to identify the part in the image. After doing so, it passes the initial pose data to the lower level. At the lower level, a controller is implemented that places the arm at the initial location. If the task was successful, the lower level will send a message to the higher level. Then the higher level tries to grip the object by using visual servoing. If it succeeds, a message will be sent to the decision unit. If by any means the task fails, the higher level chooses to repeat it or to abandon it.

Figure 10 illustrates the simplified state machine.

The decision unit receives the success messages and decides what subtask should be called in next step and what is the required information for that task.

The main advantage of such design is that for more complex tasks (such as Transportation test and technical challenge) we can break the task into simpler subtasks, each of which we have already taken care of.

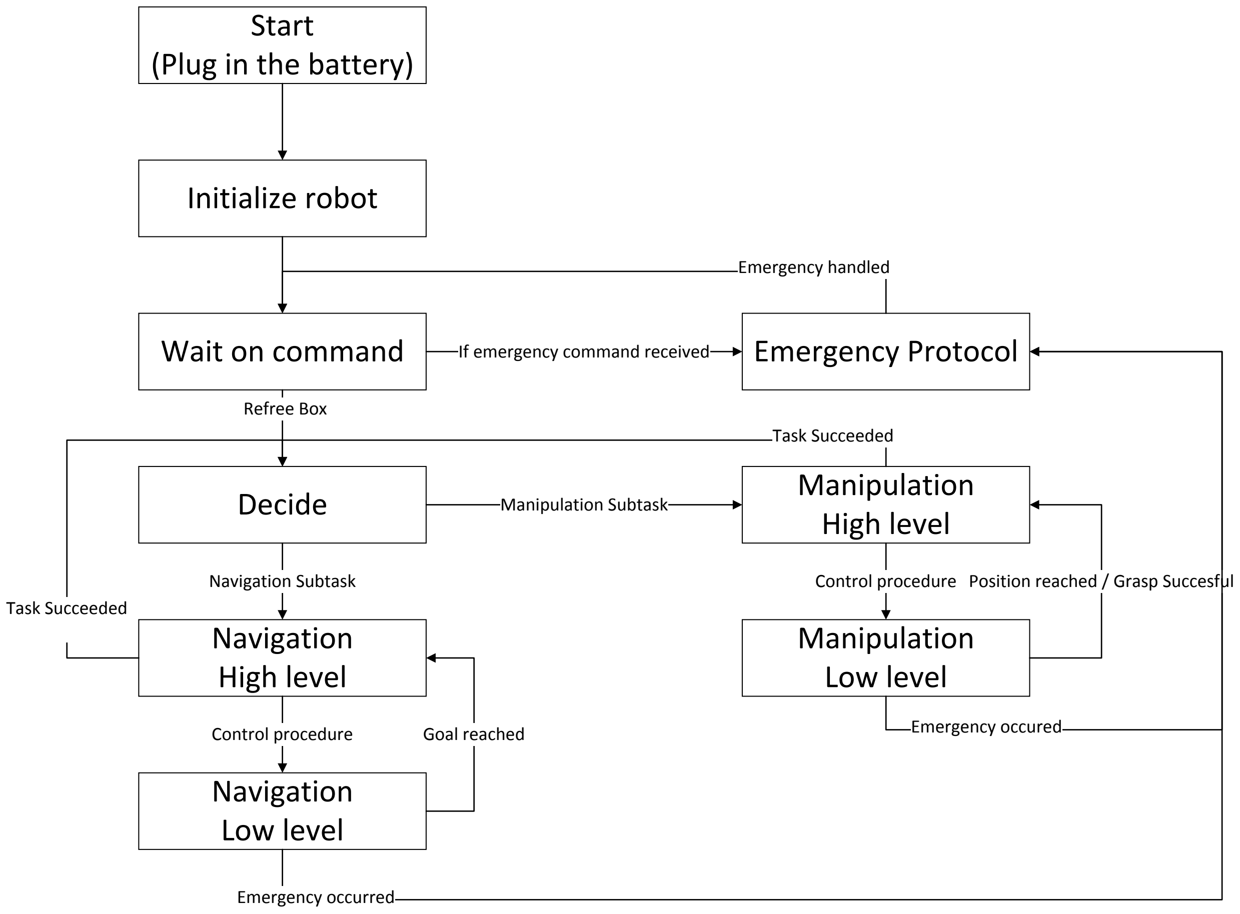


Figure 10. State Machine.

4. Reusability and Applicability

As we used ROS system in every step of our progress, we can ensure good reusability and applicability on other systems similar to ours. Additionally, most of the software is written in standalone packages that make it easy to integrate them into other systems.

5. Conclusion and Future work

In this paper, we described the current state of our team (AUT@Work). It provides detailed information about hardware (base platform and manipulator) and software (Navigation, Perception and, Manipulation). In this research, we tried to develop everything as an individual component so that they may be used as a functional module in other robots.

In future work, a complete simulation of the robot in V-REP[1] will be used to evaluate navigation and manipulation algorithms. The ARC@Work team is looking forward to participation in the RoboCup 2017 in Japan.

6. Reference

[1] E. Mehrabi, E. Babaians, A. Ahmadi, A. Sheikhjafari, S. Gharghabi, N. Khazaee, and S. Shiry Ghidary, "AUT @Home 2016 Team Description Paper".

[2] O. Diegel, A. Badve, G. Bright, J. Potgieter, and S. Tlale, "Improved Mecanum Wheel Design for Omni-directional Robots", Proc. 2002 Australasian Conference on Robotics and Automation, pp. 117-121, 2002.

[3] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation", IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR), 2011.

[4] G. Caterina and J. Soraghan, "Adaptive Template Matching Algorithm Based on SWAD for Robust Target Tracking", Electronics Letters, vol. 48, no. 5, p. 261, 2012.

[1] V-REP is a simulation tool for industrial and educational purposes. To read more about it please refer to this link: http://www.coppeliarobotics.com/