[ELE00118M] MSc Practical Robotics (PRAR)

[ELE00103M] Year 4 Robotics module

Lab Session 9 Handout

Lab 9: Robot Navigation/Assessment Guidance

Lecturer: Mark A Post (mark.post@york.ac.uk)

Technician: Mike Angus

1. Aims and Objectives

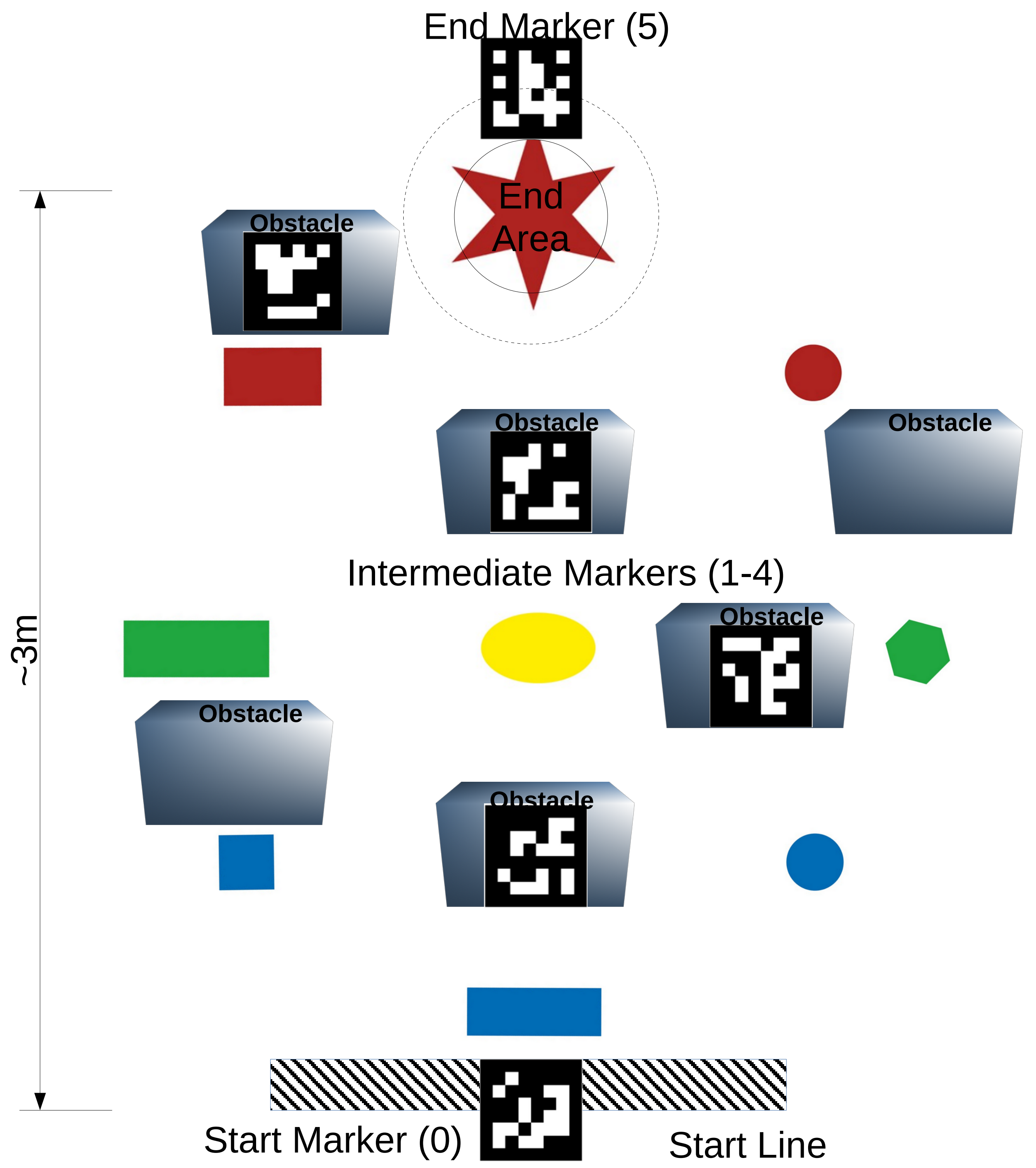
In this laboratory session, you will prepare for your assessment project by putting together navigation functions that will enable your robot to traverse an obstacle course by using its sensors.

2. Learning outcomes

- Bring together sensing, actuation, and programming expertise in one application
- Demonstrate ability to implement complex behaviours on embedded hardware
- Understand and design for the navigational challenges of real-world robotics

3. Software and hardware

You will be building on all of the software and hardware that you have constructed in the previous labs. By now you should have a completely constructed robot with a networking set up, a USB-serial link, the IR sensor calibrated and working, and the motors connected and moving reliably with feedback from the wheel encoders. Below is the recommended wiring diagram for all of this hardware on your robot's RasPiO Pro Hat. Make sure that all the parts of your robot are working as expected and ask for help if something is still not working as you would expect. You have access to an example obstacle course in the PT410 lab (shown below), and if you are working from home you can set up a simple obstacle course to test your robot's navigational abilities – the only restriction is that you need to *use your robot's sensors to make navigational decisions* ("autonomy") and not just pre-program a route for your robot to follow ("automation").



Example Obstacle Course for your Robots to use in the Module Assessment

4. Pre-Lab Preparation

- Ensure that you have completed Lab 8 and previous labs so that your robot is behaving predictably and repeatably, and all sensors and actuators are working.
- Review the lecture material on planning and navigation, and previous lectures if you have missed any so you are familiar with the methods available.
- It is not expected that you will necessarily complete all the tasks to a high standard in this lab during the time available (if you do so you will be very close to completing your work for the assessment!) Pragmatically, think about what your robot is able to do right now and make a plan for how you will accomplish basic localization, mapping, and planning based on what you have learned in the module so far, and based on the suggestions given in this lab. Then after this lab, continue to work on your robot until it is able to complete the assessment tasks.

5. Tasks

TASK 1: LOCALIZATION

Although you are not expected to implement a full Simultaneous Localization and Mapping (SLAM) solution for your assessment, you can still perform some basic localization and then a mapping step with the limited set of sensors on your robot.

The most useful sensor for localization on your robot is the camera, as you can use it with the OpenCV library to identify a wide range of different shapes and colours. First, place the provided coloured shapes or ArUco tags (or both) on surfaces near your robot. Angle the camera so you can image the objects clearly. Take the code that you worked on in Lab 6 for recognizing these objects and combine it with your movement code for your robot. You may want to multi-thread the vision code in its own function using threading in the language of your choice (e.g. pthreads, multiprocessing, etc.) or otherwise make sure it does not make your robot unresponsive for a long time by calling it infrequently or by stopping your robot first.

If you are able to recognize the shapes you put up with the camera, then try combining it with the range sensor from lab 3. If the range sensor is pointed precisely with the camera, you can find the distance to the object if the object is detected to be close to the center of the camera. Experiment with programming your robot to turn left and right in small increments to line up an object in the camera so that it is seen by the range sensor. Can you make your robot automatically line itself up with a desired object and go to it?

Finally, incorporate the wheel encoders from lab 5 to estimate how far you move when travelling to an object or between objects, keeping in mind that this is not a good estimate over long distances. You will want to use the camera and rangefinder for “long-distance” estimates of movement and the wheel encoders for small movements and turns. You can keep an odometry estimate of your robot’s movement that is different for different behaviours, e.g. wheel encoders while turning and rangefinder while approaching an obstacle. Try to make your robot travel from a start point to an obstacle (requiring a turn or maneuver), and measure the distances physically for comparison. Make your robot notify you when it has “reached” an obstacle within a nominal distance – this is needed for planning. What combination of sensors gives the most accurate estimation of your robot’s movement?

TASK 2: MAPPING

With some degree of localization, you can now consider making a rudimentary map of the area your robot traverses. There are two common map type choices: making a grid-based map, or making a graph-based map. In either case, you can also incorporate prior information into your map that is used by your robot, for example by measuring the physical locations of the start and end locations and obstacles on your course, and then using them as landmarks to be identified by your vision system. When your robot identifies a visual landmark, you can assume that your robot is in front of the landmark at a distance that you can measure with your IR sensor. You will also need to specify where in your map your robot starts – if your robot starts at the zero origin in your map coordinate system then make sure your estimated landmarks are measured from there.

The form of the map could be simply an array of values - two dimensional arrays make good grid maps, linked lists make good graph maps. You could also use a dictionary in Python to associate landmarks with positions, and therefore with vectors to other landmarks. Try creating one of these, and have your robot enter landmarks it recognizes into its map as it recognizes them. You can use the estimated position of your robot itself as the location of the landmark, but try to refine this by using the camera and rangefinder to estimate how far your robot is from the object, and in what direction.

TASK 3: PLANNING

Once you have a rudamentary map, try programming a simple planner that will make your robot travel to a sequence of landmarks in your map if you have pre-defined a map. Alternately, if you are adding objects to your map as you go, you can have your robot simply travel forward toward the goal, avoid obstacles, and track where landmarks are.

Once you have a map with landmarks at known locations, program your robot to travel to the landmarks in sequence so that you can refine its movement and obstacle avoidance. You can easily define a path as a graph sequence, to travel in a straight line between coordinates of obstacles, or a grid sequence, to travel sequentially through a set of grid coordinates. In both cases you need to the planner to use your localization functions to estimate whether your robot has reached a given location or "waypoint" before instructing it to move to the next one. Start with a static path that you want the robot to follow, and make sure that the vision, encoders, and range sensor are all working as desired and that the robot is both following the path and avoiding obstacles on the way. Using the robot's kinematics and inverse kinematics with your motor control functions from Lab 4, you should be able to set the motors to follow a rudamentary path from the end location back to the start location, even if it is just choosing a direction to the nearest recognized landmark in your map.

Finally, if you have everything else working, try to find the most efficient sequence of landmarks or grid locations to use to travel between the start and end locations in your map. You can use any algorithm you like but for small maps Dijkstra's algorithm or A* works fast enough to give the best answer consistently. For debugging purposes you may want to print out your map in a text or graphical format while you do this.

5. Hints

- Don't try to make perfect maps, make functional maps that give usable directions, and rely on your robot's obstacle avoidance and vision to reach the goal
- Markers on obstacles are easy to see and could be used for a basic pose graph
- Shapes could be detected with blob detection on floor
- Camera can be aimed at an angle to view markers on the floor
- Colours are easy to detect and could be used to aid position estimation
- Range sensors to avoid objects and supplement wheel encoders
- Potential solutions include:
  - Start out facing forward and circle around obstacles until you identify the end
  - Identify each ArUco tag in sequence, and then move to search for next one
  - Build a pose graph from each shape or ArUco tag visited and then re-trace
  - Search randomly and follow colours blue-green-red and then back
  - Follow walls to the left or right with range sensors to the end area
- You are welcome to try out different kinds of courses depending on the navigation algorithm you are developing - you are encouraged to customize the course you choose to show of how your navigation works!