



# ME-348/ME-392Q.9

## Advanced Mechatronics Systems

### HomeWork & Laboratory

---

INSTRUCTOR: Benito R. Fernández<sup>1</sup>, Ph.D.  
ASSIGNED: 2017.04.11   
DUE: 2017.04.27 

---

# HUMERS

## Heterogeneous Unmanned Mobile Evolving Robotic System

---

### Description

For our final homework, we are going to look at how to control a set of heterogenous robots (with different characteristics and sensors/actuators).

You will apply the HUMERS to a robot-maze problem.

As before, please turn in any code (soft copy) that you use for the homework.

## Evolutionary Robotics

The system in question is a very simple maze-robot problem. A diagram of the maze is shown in Fig. 1, where walls, start, goal, and food are drawn. The objective is to develop a mechatronic device (robot and controller software) by which the robot reaches the goal. In your algorithm you are encouraged to apply the concepts of OOP (Object-Oriented Programming) to develop your control algorithm. The robot may or may not know the size of the World.

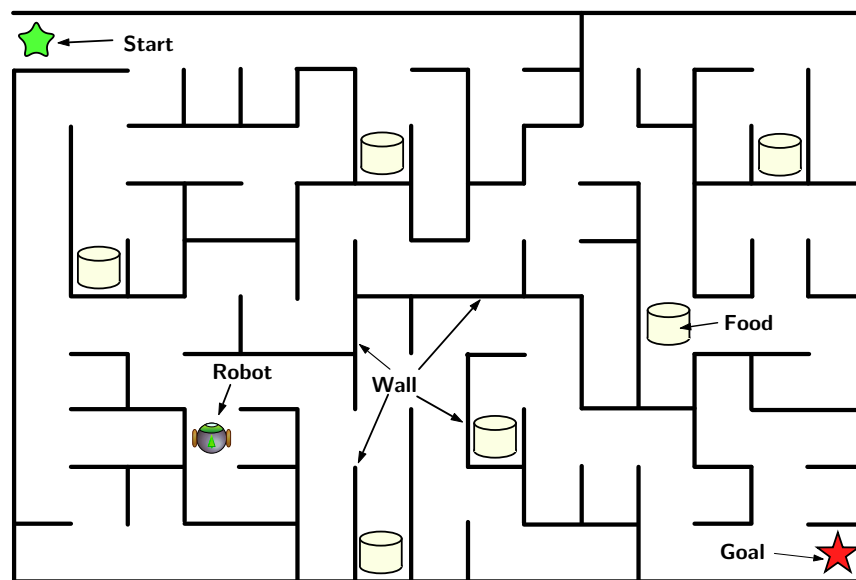


Figure 1: Maze showing the robot facing north at coordinates  $(x, y) = (8, 4)$ . The starting point is assumed to be at  $(1, 1)$  and simulation terminates when the robot gets to the opposite corner of the maze at  $(10, 15)$ . The maze has walls that the robot cannot penetrate. There are also bits of “food” (realized in the physical maze with gray squares on the ground) that the robot can “eat” to replenish energy.

### The Environment: Robot-Maze

The code for the original “Evolutionary Robotics” maze is provided as an attachment in UTBox (You don’t have to use it. Is there just for reference). The robot can navigate the maze and it is assumed that there is a reinforcement given to the robot,  $R(\mathbf{z}(t), \mathbf{u}(t), t)$ , based on its *action*,  $\mathbf{u}(t)$ , and location within the maze (*state*),  $\mathbf{z}(t)$ . Where the state vector is represented by  $\mathbf{z} = \{x, y, \theta\}$ , where  $(x, y)$  are the row (vertical) and column (horizontal) coordinates of the robot and  $\theta$  is the orientation/heading of the robot. The top-left corner of the maze,  $(x, y) = (0, 0)$  is the starting point (start). In the original code, the heading was a multiple of  $\pi/2$ , that is equivalent to  $heading \in \{\text{East, North, West, South}\}$ . In your

code, you are going to have a continuous value for  $\theta$  (you might be able to discretize it in degrees depending on the wheel encoders).

The robot has limited motion capabilities. It has two wheels that can move in both directions that are bounded between a  $\omega_{\min} < \omega_i < \omega_{\max}$ , where  $i = \{R, L\}$  represent the right and left wheels rotational speeds. For simplicity we can assume that  $\omega_{\min} = -\omega_{\max} = \Omega$ . The angular speed is controlled by sending a pulse width modulated signal, with a pulse width,  $p$  between  $1.0 < p < 2.0[\text{ms}]$  and a through of  $20.0[\text{ms}]$ . In reality, the mapping between pulses  $p_i$  and angular velocity,  $\omega_i$  is not linear and tends to saturate at  $1.25[\text{ms}]$  and  $1.75[\text{ms}]$ . In general the servo motors are calibrated so they do not rotate when the pulse width is  $p_0 = 1.5[\text{ms}]$ . Hence, the angular speed is related to  $p$ :

$$\omega_i = (p_i - p_0) \times \omega_{\max}$$

Given the radius of the wheels,  $R_w$ , we can compute the each wheel's (forward) speed,  $v_i$ :

$$\begin{aligned} v_R &= R_w \times \omega_R = R_w \times (p_R - p_0) \times \omega_{\max} \\ v_L &= -R_w \times \omega_L = -R_w \times (p_L - p_0) \times \omega_{\max} \end{aligned}$$

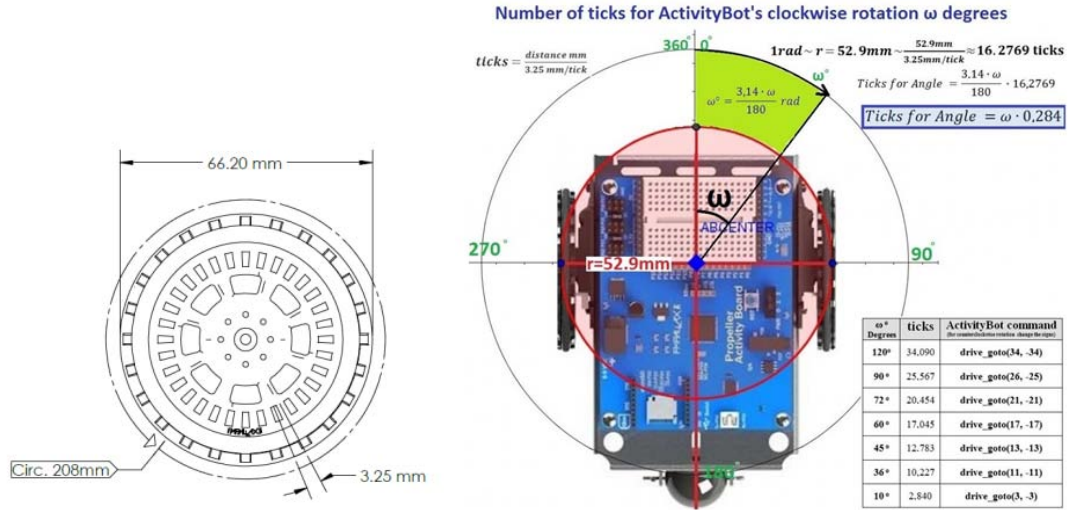


Figure 2: PARALLAX ActivityBot wheels with rotational encoder.

The command to move forward requires the right wheel to rotate clockwise at a given speed,  $\omega_R = \omega$  and the left wheel to rotate at the same speed counterclockwise,  $\omega_L = -\omega$ . Given the radius of the wheels,  $R_w$ , we can compute the car's (forward) speed,  $v_C$ , and the

car's angular (rotational) speed,  $\omega_C$  as a function of left and right wheels angular velocities:

$$v_C = \frac{1}{2}(v_R + v_L)$$

$$\omega_C = \frac{1}{2}(v_R - v_L) \times \frac{W}{2}$$

where,  $W$  is the wheel span (between left and right wheels). You can also use the wheel decoder to measure displacement of the wheel.

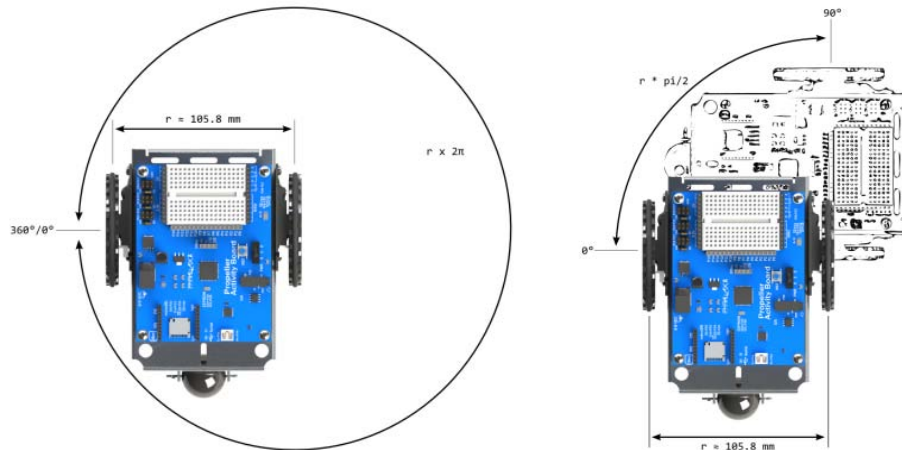


Figure 3: PARALLAX ActivityBot calculation of number of ticks for 360 and 90 degrees. The rotation uses only the left wheel while the right wheel is fixed.

Given the radius of the wheel and the spacing of the ticks (see Fig. 2), the wheel moves at  $\lambda = 0.325[\text{mm}/\text{tick}]$ . For details on how to calibrate rotation and translation, refer to [calibrate-your-activitybot](http://learn.parallax.com/activitybot/calibrate-your-activitybot)<sup>2</sup>, [regular-polygons-and-circles](http://learn.parallax.com/activitybot/regular-polygons-and-circles)<sup>3</sup>, and [go-certain-distances](http://learn.parallax.com/activitybot/go-certain-distances)<sup>4</sup>. For navigation, refer to [navigation-basics](http://learn.parallax.com/activitybot/navigation-basics)<sup>5</sup>, [roaming-infrared-flashlights](http://learn.parallax.com/activitybot/roaming-infrared-flashlights)<sup>6</sup>, and [navigate-ultrasound](http://learn.parallax.com/activitybot/navigate-ultrasound)<sup>7</sup>.

In the UTBox folder for the homework, you can find files and code for most of the components of the propBot.

<sup>2</sup><http://learn.parallax.com/activitybot/calibrate-your-activitybot>

<sup>3</sup><http://learn.parallax.com/activitybot/regular-polygons-and-circles>

<sup>4</sup><http://learn.parallax.com/activitybot/go-certain-distances>

<sup>5</sup><http://learn.parallax.com/activitybot/navigation-basics>

<sup>6</sup><http://learn.parallax.com/activitybot/roaming-infrared-flashlights>

<sup>7</sup><http://learn.parallax.com/activitybot/navigate-ultrasound>

## Your Work

- ❶ Design and build a mobile robot that can navigate the maze. Suggested sensors to use are: the Sonar/IR (PING/IR) turret, to detect walls; IR Wheel Encoders, to perform odometry; QTI (IR) sensors at the front-bottom, to be able to detect food and danger lines on the floor; and optionally, IR sensors on the sides, to be able to follow a wall; Compass<sup>8</sup>, to detect heading or direction of the robot body.
- ❷ Implement a controller that Wanders on the maze Avoiding hitting obstacles (including walls). This should include avoiding crossing the black lines (danger zone) detected by the QTIs – They should be treated as walls. Since you are using the QTIs, count the number of gray squares (food) found.
- ❸ Write a logger object that keeps count of the pulses of the wheel encoders with a time-stamp for odometry estimating the current location of the car (including orientation). With this information, you would be able to create a path of the robot. Also measure and record the data of the IR sensors and Sonar. It is suggested that you measure the distance of the sonar/IR turret at 3 or 5 angles (from the front of the car, e.g.  $\{0, \pm\pi/4, \pm\pi/2\}$ ). You may store all the data (with time stamp) in a file in the microSD card. With this information, you should be able to create the maze map.
- ❹ Devise code to use the XBee RF module socket to communicate with your PC and transfer the data. First, the data file from the SD card; second, the data of the sonar, IR, and encoders as they are generated. You may write a VI in LabVIEW that takes the data and displays it on the screen as is received from the PropBot.
- ❺ You should be able to create a Map of the maze as the data comes in and update the position of the robot in the maze (SLAM: Simultaneous Localization and Mapping). Create a LabVIEW program that uses the data recorded and create a Map of the environment (maze). Assume that all the walls are at right angles (for dead reckoning). Identify in your drawing the hazards (danger zone lines) and food locations.
- ❻ Write or modify your main program to take GO commands (from the screen in your VI Front Panel or your PC's keyboard) where you specify where in the maze you want your robot to go.

---

<sup>8</sup>There is a way of mounting the compass in the sonar turret that allows to obtaining the direction of the measurement and heading.