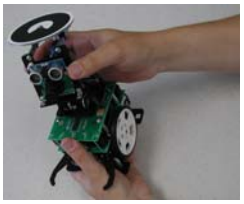
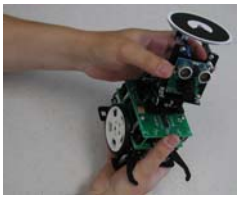


COMP4807 - LAB #2
Position Estimation
(Due Sun. Oct. 14, 2012 @ 11:59pm)



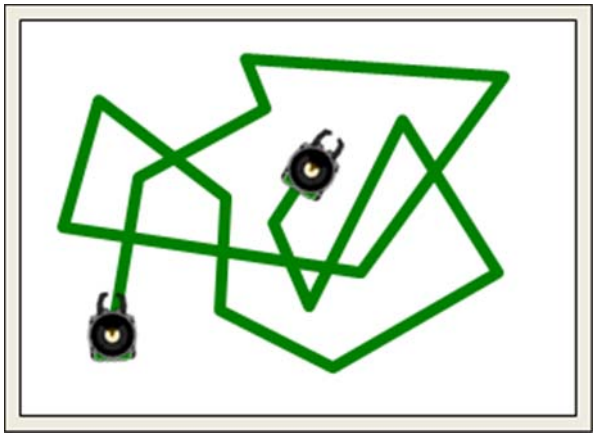
In this assignment you will become familiar with the issues involved with estimating the position of the robot through use of the robot's digital encoders to perform dead-reckoning. Also, you will learn how to send data from the robot to the PC through the RobotTracker software and **Bluetooth** devices.

Robot Setup:

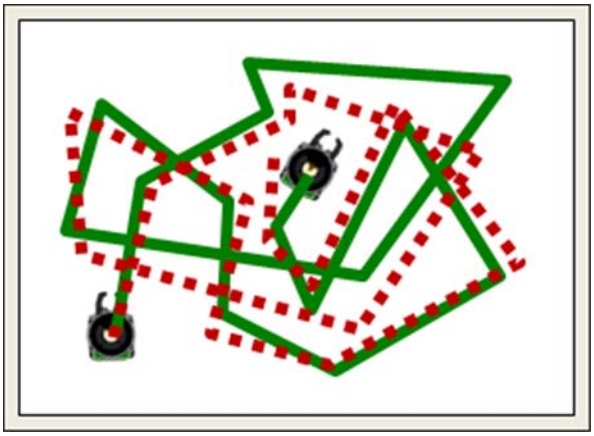
Make sure the robot is turned **off** and then turn **on** (i.e., down position) dip switch **1** (**F_IR**) and **3** (**BLU**) from the 8-position switch on the top of the robot and turn **off** (i.e., up) all other switches (i.e., **2, 4, 5, 6, 7** and **8**). At the back of the robot, make sure that the 2-position dipswitch is set so that switch **1** (**Block**) is up and switch **2** (**Encode**) is down.

The Fun Stuff:

1. Copy your **Wander.spin** code from LAB1 to a new file called **temp.spin**. Adjust the **temp.spin** code so that the robot only moves forward and spins left and right (i.e., no turning arcs). Ensure that your motors are aligned properly by using the stopped motor values from part 1. Also, adjust your ramping code to make sure that the robot does not attempt to spin left or right until its current servo speed values have reached zero. This will allow the robot to make a complete stop after each forward movement and after each spin movement. Now insert code that allows the robot to estimate its own pose based solely on the wheel encoders. You can calculate the pose as often as you would like, but make sure that you re-compute the pose after each time the robot has changed servo speeds (i.e., during each of the “stopped” intervals). You will not be able to properly verify your code until you complete the next part.

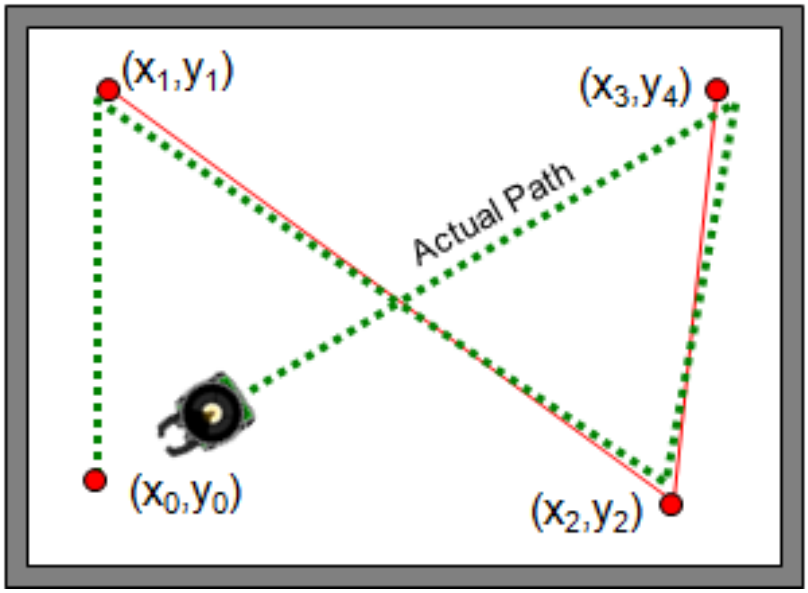


2. Carefully, read over the RobotTracker examples at the end of Chapter 3, and then create a new file called **EstimatePosition.spin**. Now you will be using the bluetooth device to send the estimated positions to the PC for debugging/storage. Modify the code so that the robot receives the initial **(x, y)** position and angle of the robot upon startup and then repeatedly sends back the estimated **(x, y)** position and estimated **angle**. You will need to create your own Planner class (call it **EstimatePositionPlanner.java**) ... see chapter



3 notes to get more information on how to use the planner. Upon startup, the planner will automatically create a trace file called which contains the header line **x, y, angle**. Each time the robot sends back its estimated position, you should store the estimated (x, y, angle) in the trace file. Note that each time you start a new trace, a new trace file will be created. You will also need to send the estimated position to from your planner code to the **RobotTracker**. You will need to make use of the Planner method called: sendEstimatedPositionToTracker(**int** x, **int** y, **int** angle) which will allow you to display the estimated position on the screen. Make sure that the **Show Estimated Path** option is enabled in the RobotTracker's **View** menu so that you can see your estimated position. Using the **RobotTracker**, record a 3 minute **.avi** video (called **EstimatePosition.avi**) of the robot wandering around in the environment. Rename the trace file to **EstimatePosition.trc**. Take 3 screen snapshots ... showing each trace separately and one showing both overlayed together (name these files **ActualPosition.png**, **EstimatePosition.png** and **OverlayedPosition.png**). Make sure to set path colors that are clearly visible with high contrast when compared to the background.

3. Copy the **EstimatePosition.spin** code to a file called **InverseKinematics.spin** and modify that code to accept an initial pose (x_0, y_0, angle) from the RobotTracker as well as 3 additional user-defined points (x_1,y_1), (x_2,y_2) and (x_3,y_3). Upon receiving these points, the robot should attempt to automatically navigate from point (x_0,y_0) to (x_1,y_1) to (x_2,y_2) to (x_3,y_3) and then back to (x_0,y_0) and stop. It must do this only using the encoders (i.e., without any pose feedback from the Planner).



Your code should make use of the wheel encoders and attempt to travel in a straight line between points, while turning at each point to an appropriate angle. Your code need not send any particular data to the PC. You will need to write a new **InverseKinematicsPlanner.java** planner to send the initial pose and user-defined points. Using the **RobotTracker**, record an **.avi** video (called **InverseKinematics.avi**) of the robot moving along a desired path in the environment. Allow the user to select the desired path from the **RobotTracker** (using the Plot feature to create 4 points) which is then sent to the robot (i.e., by calling the planner's **sendDataToRobot(**int**[] data)** method exactly once ... sending all 4 locations (and initial starting angle). Make sure to also save a snapshot (called **InverseKinematics.png**) showing the user-defined path (as you entered it) as well as the actual path that the robot took. Again, make sure to set path colors that are clearly visible with high contrast when compared to the background.

Submission:

Create a webpage for this course that contains a simple report on your lab (i.e., a description of what you did and what each part of the assignment was trying to do). Include downloadable links for all of your code as well as any snapshots, trace files and videos. There is no need to make the webpage beautiful … but that's up to you. Login to Carleton's WebCT system and submit a link to your website. Make sure that the website is up and running at least until your code is marked. If you have never created an HTML page before, you can use Mozilla's **Sea Monkey** browser which has a built in editor that allows you to make simple pages. Always keep a backup of all your work (perhaps on a USB key or burn a CD). Here is a summary of what to hand in:

JAVA CODE:

- `EstimatePositionPlanner.java`
- `InverseKinematicsPlanner.java`

SPIN CODE:

- `EstimatePosition.spin`
- `InverseKinematics.spin`

SCREEN SNAPSHOTS:

- `ActualPosition.png`
- `EstimatePosition.png`
- `OverlayedPosition.png`
- `InverseKinematics.png`

VIDEO CLIPS:

- `EstimatePosition.avi`
- `InverseKinematics.avi`

TRACE FILES:

- `EstimatePosition.trc`