Assignment 1: Control and obstacle avoidance

ETH Zurich

Due: Thursday, 24.10.2013 at 13:00

It's time to put your new robot into use. You would like your robot to go to places as you command, such as fetching beer from the refrigerator. Unfortunately, your apartment is not empty. You don't want to have to tell your robot how to avoid every wall and furniture that may lay in the robot's way; instead, you would like your robot to use its sensor to avoid them as it sees necessary.

1 Robot control

1.1 Background

The main objective of robot control is to calculate solutions for the proper corrective action from the controller that result in system stability, that is, once the system reaches the target point, the system will stay within a certain distance from the target point without oscillating around it. Control algorithms can be open loop or closed loop (feedback). In open loop control, you control the robot without any feedback from the internal or external sensors. Feedback control, on the other hand, uses internal or external sensing to determine the current error between the actual and desired state of the robot.

Proportional-Integral-Derivative (PID) control [1] is a popular feedback control design. The proportional term is proportional to the error between the desired and actual system outputs and controls how quickly the robot reacts to the error. The integral term is proportional to both the magnitude of the error and the duration of the error and accelerates the movement of the process towards the desired output and eliminates the residual steady-state error. The derivative term is proportional to the slope of the error over time and improves settling time and stability of the robot.

Mathematically, the PID controller is

$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, \mathrm{d}\tau + K_d \frac{\mathrm{d}}{\mathrm{d}t} e(t) \tag{1}$$

where $u(t)$ is the control output; K_p, K_i, and K_d are control gains for the proportional, integral, and derivative terms; e is the error between the desired value and measured value; t is the current time; and, τ is the total time from time 0 to the current time t.

We can make a robot go to a desired position(goal) by controlling its heading θ_{robot} towards the goal, as shown in Figure 1. Given the robot's position (x_r, y_r) at time t, the heading error θ_{error} between the robot and the goal position (x_g, y_g) is

$$\theta_{error} = \theta_{goal} - \theta_{robot} = \arctan(\frac{y_g - y_r}{x_g - x_r}) - \theta_{robot}. \tag{2}$$

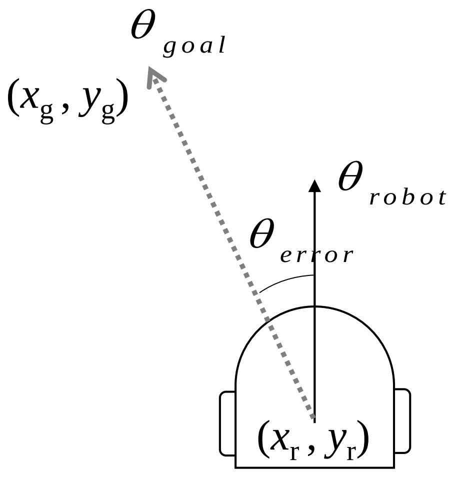

Figure 1: Control output θ

1.2 Task

Write a PID controller for Thymio II in Roboscoop to control its heading. Your code should take desired goal position as input and control the heading towards the goal using the PID controller.

1.2.1 Hints

- Robot odometry: `roboscoop_thymio_navigation_driver.py` publishes robot's odometry at time t and sends velocity command (v_x, v_z) to the robot. Your job is to send appropriate velocity commands based on the robot's odometry.
- Goal tolerance: Consider including a threshold such that the robot stops moving when it is within a threshold of the goal.
- Linear velocity: Consider making the linear velocity depend on the angular velocity. This will make the robot slow down when it needs to make a big turn and have a smaller turning radius.

2 Obstacle avoidance

2.1 Background

Obstacle avoidance is the process of satisfying a control objective without colliding into obstacles. Obstacle avoidance should be written such that in the case of obstacles, the robot makes appropriate motions around the obstacles while trying to achieve the goal following the shortest path. If there are no obstacles, the robot should move directly towards the goal location.

TangentBug [2] is an obstacle avoidance algorithm in the Bug [3] algorithm family. The Bug algorithms combine local planning algorithms with global planning algorithm. With a minimal introduction of a global model, the Bug algorithms ensure that the purely reactive local planning algorithms can converge to the desired goal globally. TangentBug is a Bug algorithm, specifically-designed for range data. The basic idea behind the algorithm is as follows:

1. Move toward the goal T until:
    - If the goal T is reached, stop.

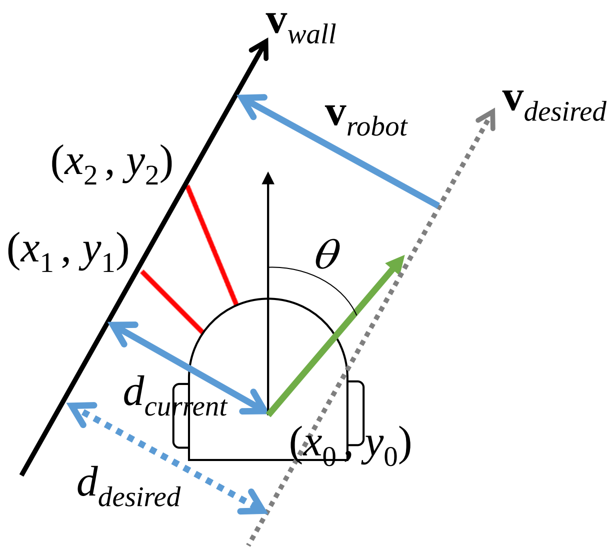

Figure 2: Control output θ for boundary following

- If an obstacle in the direction towards to goal is detected, go to step 2.

2. Choose a boundary-following direction and move along the obstacle while recording the minimum distance $d_{min}(T)$ to the goal T until:
   - The goal is reached. Stop.
   - If the leave condition holds, i.e., the robot see no obstacle at V_{leave} such that $d(V_{leave}, T) < d_{min}(T)$, go to step 3.
   - If the robot has completed a loop around the obstacle, stop and report that the target is unreachable.
3. Perform the transition phase. Move towards V_{leave} until reaching a point Z such that $d(Z, T) < d_{min}(T)$, then go to step 1.

In boundary-following, the goal is to keep robot's heading parallel to the wall while at the same time keeping the robot a constant distance away from the wall. Figure 2 shows the basic concept behind boundary following. Taking (x_1, y_1) and (x_2, y_2) as the two closest sensor values to the robot, we can estimate the closest wall $\mathbf{v}_{wall}$ as

$$\mathbf{v}_{wall} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}. \tag{3}$$

The vector perpendicular to the wall is then

$$\mathbf{v}_{robot} = \begin{pmatrix} y_2 - y_1 \\ -(x_2 - x_1) \end{pmatrix}. \tag{4}$$

We can now calculate the distance $d_{current}$ from the robot to the wall $\mathbf{v}_{wall}$ as a projection of the vector $\mathbf{v}_{sensor} = \begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \end{pmatrix}$ from the robot origin (x_0, y_0) to a point (x_1, x_2) on the wall $\mathbf{v}_{wall}$ to a unit vector $\hat{\mathbf{v}}_{robot}$, i.e.,

$$d_{current} = |\hat{\mathbf{v}}_{robot} \cdot \mathbf{v}_{sensor}| = \frac{(y_2 - y_1)(x_1 - x_0) - (x_2 - x_1)(y_1 - y_0)}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}. \tag{5}$$

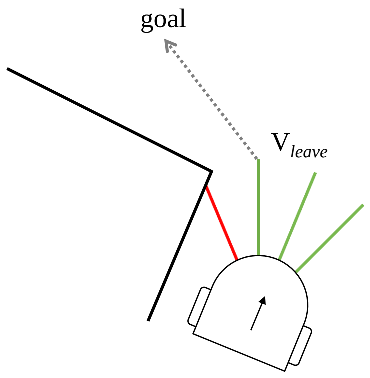

Figure 3: Transitioning to the V_{leave}

Given the desired distance $d_{desired}$ between the wall and the robot, we can now compute the heading θ for the robot from $\hat{\mathbf{v}}_{wall}$ and $\hat{\mathbf{v}}_{robot}$ as

$$\theta = \arctan(\frac{\mathbf{v}_{\theta,y}}{\mathbf{v}_{\theta,x}}) - \theta_{robot}, \tag{6}$$

where

$$\mathbf{v}_{\theta} = d_{desired}\hat{\mathbf{v}}_{wall} + (d_{current} - d_{desired})\hat{\mathbf{v}}_{robot} \tag{7}$$

Note that if the vectors are computed with respect to the robot's coordinate frame, then θ_{robot} is zero.

At some point, there is no more obstacle on the robot's way to the goal. The location in which the robot can go out of the obstacle avoidance is V_{leave} as shown in Figure 3. This exit point V_{leave} is the first free space that ensures that the robot is closer to the goal than it has been, i.e., $d(V_{leave}, T) < d_{min}(T)$. Once the robot moves to this exit position V_{leave}, it will be closer to the goal.

2.2 Task

Write TangentBug obstacle avoidance algorithm in Roboscoop. The TangentBug algorithm requires three distinctive behaviors: going to the goal, avoid obstacle (following a wall), and transitioning from obstacle avoidance to going to the goal. Optionally, implement an aseba message to change robot's color and use the message to indicate the robot's current state/behavior through its color.

2.3 Hints

Thymio's range sensors are limited in number and range. Consider the following modifications to the original TangentBug algorithm.

- **Sensor calibration**: Be sure to calibrate properly all the sensors before starting the controller implementation.
- **Obstacle detection**: The original algorithm is designed for a point robot, but Thymio has volume. A minimum of three sensors must be unblocked in the direction of travel for Thymio to travel safely.
- **Obstacle avoidance**: In the obstacle avoidance mode select different distance thresholds for each sensor (when detecting an obstacle) or a unique threshold (if it exists) that you are sure is inside the working range of all the sensors. This is because the detection range

for one sensor can be much lower than the others. Therefore, you must always ensure that all the thresholds you set/use for the control are inside the working range of all the sensors that you use to control the device. Otherwise, if the controller uses a threshold that is outside the range of one/more sensors, then the controller is going to behave in crazy ways and the controller would be unstable or not robust.

- **Unreachable goal**: Loop closure is a difficult problem and requires tracking of robot's trajectory and good sensory information. Consider a simple solution in which your algorithm detects when the robot comes near the starting point of obstacle avoidance after it has been significantly away from it.

3 Grading

3.1 In-class demonstration (20 points)

On Thursday, 24.10.2013, during the exercise session, your implementation will be tested on the following two tasks:

1. Go to goal (10 points)
2. Go to goal while avoiding an obstacle (10 points)

For each task, you will be given a point (x, y) between $(-1m, -1m)$ and $(1m, 1m)$ as the goal and a linear velocity between $0.05m/s$ and $0.1m/s$.

The grading scheme for each task is as follows:

	Accuracy	Restart	Bumping	State indicator
Task 1			-	-
Task 2				

- Accuracy: Position control to the goal
  - Error of more than 30%: 0 point
  - Error of more than 27% up to 30%: 1 point
  - Error of more than 24% up to 27%: 2 points
  - For every 3% reduction of error: +1 point
  - Error of 3% or less: 10 points
  - Example: Given $(0.5, 0)$ as the goal, a robot that arrives at $(0.43, 0)$ as the goal will receive 6 points.
- Restart: You can restart the robot up to three times.
  - Once: no penalty
  - Two to three times: -2 points
- Bumping into the obstacle (for task 2 only)
  - **Bonus point**: No bumping: +1 points
  - One to two times: no penalty
  - Three to four times: -1 pt
  - Five or more: -2 points
- **Bonus points**: change robot's color to indicate its state: +2 points (for task 2 only)

  - Yellow for going to goal
  - Red for boundary following
  - Blue for transitioning to the goal
  - Green when the goal is reached

3.2 Software quality (20 points)

On the due date at the due time (Thursday, 24.10.2013, 13:00), we will collect your code through your svn repository. Every file that should be considered for grading must be in the repository at that time. If you have not yet set up your svn repository yet, please set it up now. The instruction for setting up the repository is in assignment 0.

- Choice of abstraction and relations (6 points)
- Correctness of implementation (8 points)
- Extendibility and reusability (4 points)
- Comments and documentation, including ”README” (2 points)

References

[1] Astrom, K., and Murray, R. 2008. Chapter 10: PID Control. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press.

[2] Kamon, I., Rimon, E., and Rivlin, E. 1998. TangentBug: A Range-Sensor-Based Navigation Algorithm. *The International Journal of Robotics Research*. 17(9):934-953.

[3] Lumelsky, V. J., and Stepanov, A. A. 1897. Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape. *Algorithmica*. 2:403-430.