Middle East Technical University   Department of Computer Engineering

CENG 460

Introduction to Robotics

Fall 2019-2020

Homework 2 - Mobile Robots

Due date: 26 11 2019, Tuesday, 23:59

1 Theory (40pts)

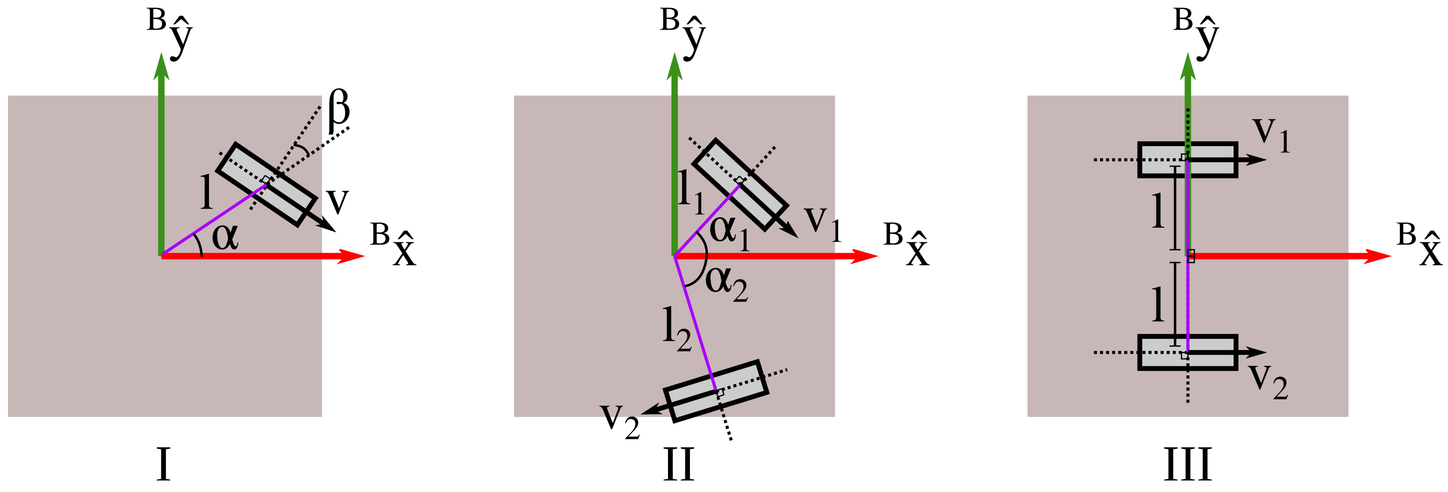

Figure 1: A top-down look at the fixed wheel configurations for a mobile robot. The unit vectors $^B\hat{x}$, $^B\hat{y}$ of the body frame are drawn on the center of the robot. The orientation θ with respect to the global frame is not shown in the drawings for simplicity.

(a) **Fixed Wheel Constraints (5pts)** Write down the 2 constraints imposed by a single wheel with radius r and angular velocity ω that has velocity of magnitude $v = r\omega$ at its center point, (Figure 1-I), in the form $f(^B\dot{x}, ^B\dot{y}, \dot{\theta}, v, \alpha, \beta, l) = 0$.

- The wheel experiences a pure rolling motion.
- The wheel can not slip sideways.

(b) **Silly Design (10pts)** 2 wheels are attached to a robot shown in Figure 1-II, with $\beta_1 = \beta_2 = 0$ and $\alpha_1 + \alpha_2 \neq n\pi,\ n \in \mathbf{Z}$. Write the 4 constraints the 2 wheels collectively impose, and figure out the sets of possible $[v_1,\ v_2]$ and $[^B\dot{x},\ ^B\dot{y},\ \dot{\theta}]$ that do not violate these constraints.

(c) **Differential Drive (10pts)** With the additional design choice of $\alpha_1 = \frac{\pi}{2}$ and $\alpha_2 = -\frac{\pi}{2}$ the robot can move more freely. For simplification, let $l_1 = l_2 = l$ as well (Figure 1-III). Apply the same procedure in (b) and find the sets of possible $[v_1,\ v_2]$ and $[{}^B\dot{x},\ {}^B\dot{y},\ \dot{\theta}]$ that do not violate these constraints. Note that the direction of v_2 is switched for convenience.

(d) **Pons Asinorum (15pts)** Solve the problem in (b), but this time $\beta_1,\ \beta_2$ are not necessarily zero (*Hint & Simplification: Use the concept of instantaneous center of rotation. If you are going to solve the problem this way, you do not need to explicitly calculate the location of ICR and the angles, distances it makes with other points, vectors etc. However you need to at least verbally/visually describe the way you would find these values.*).

2 Programming (60pts)

In this part, you are going simulate a a differential drive with Proportional Control. The formulation is available in your textbook at p.102-106. You will also need to configure `scipy.integrate.solve_ivp` Runge-Kutta solver so that you can integrate the velocities of a differential drive into its global trajectory. Stub files will be provided to you for a quick start. Please do not forget to include the following information in your plots (that are to be attached to your answers for the Theory part):

- Axis labels.
- Indicators regarding the orientation of the robot. (Robot shapes, quivers, etc.)
- A title including the gain values.
- Goal point(s) that the robot would be heading toward.
- A Legend, if multiple plots are present in a single figure.
- Comments if necessary, regarding why the solution behaves that way.

2.1 Task 1 - Moving To a Point (30pts)

Configure the numeric solver so that the differential drive moves toward a point as described in your textbook at p. 102. After you successfully implement the stub:

- Include a plot with the differential drive successfully converging to a point.
- Increase K_{pv} and plot a case that the differential drive diverges.
- Can there be a case where the differential drive is in a *limit cycle* (orbits around the point forever)? Plot this case if it exists and prove its validity mathematically. You may want to increase tolerance values. Now play with K_{pv} and report the interval the solution converges.

2.2 Task 2 - Moving With a Trajectory (30pts)

In this part, you need to halt the numeric solver when the robot is closer to the point than a certain value, and then reconfigure the solver so that you would move to the next point. After you successfully implement the stub:

- Include a plot with the values in the stub.
- Include a plot with K_{pv} increased. What is the problem here?
- Include a plot with less number of points (play with `NUM_INTERPOLATIONS`).

Implementation Specifications

If the solver terminates with an event function `t_event` will probably not be the last element of `bunch.t`. Do not mind this, if you are following a trajectory, just insert the last element of `bunch.y` as the initial value and restart the solver with unevaluated times of `t_eval` as the new `t_eval`.

Implement the functions in the given stub file with Python3. You may use libraries other than NumPy, SciPy, and Matplotlib, but you need to get permission from the course assistant first (preferably by opening a topic at the ODTUCLASS forum). The code will be reviewed with white-box technique, therefore please **comment** your intents within the hard-to-understand parts of your code.

3 Submission

You can submit handwritten solutions for the Theory part, however they need to be scanned (to pdf format), **and the hardcopies should be delivered to B202 within the next day after submission**. Submit all your files, including the scans and the filled in stubs within a zipfile named `eXXXXXXX_hw2_460_2019f.zip` to ODTUCLASS.

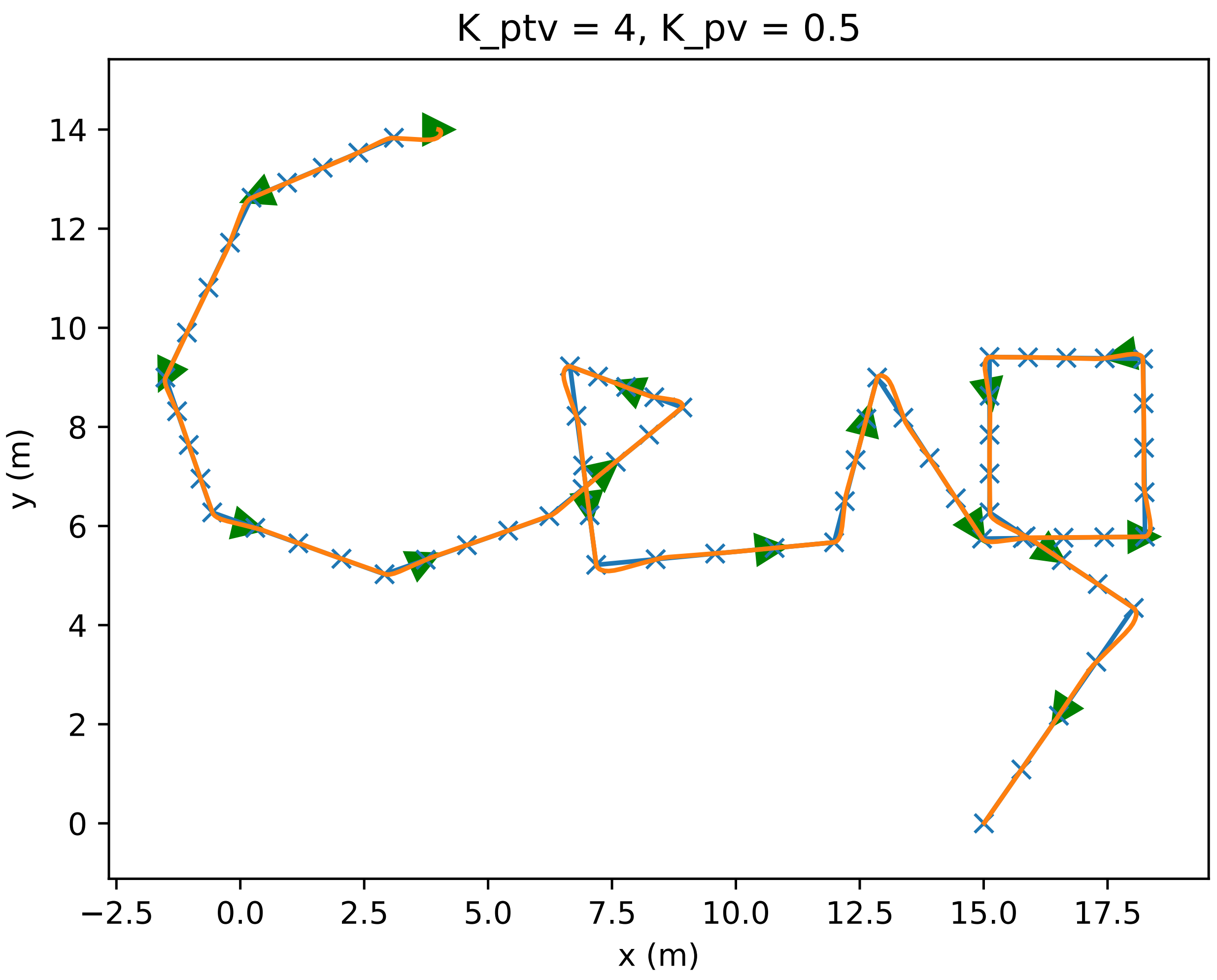

Figure 2: A Sneak Peek of the Programming Part.