# Localization Challenge

## 1. Introduction

Thanks for agreeing to spend some time hacking on robot localization with us. In this challenge, you will write a feature-based localization engine to estimate the position of a mobile robot on a computer simulated 2D soccer field. This challenge is inspired by the international RoboCup competition, where teams of autonomous robots play against each other in the game of soccer. We will evaluate your solution based on accuracy, code quality, and how clearly you describe your technique.

## 2. Landmark-Based Localization

### 2.1 Problem Statement

The robot state is represented by a three-dimensional vector indicating the robot's position and orientation on the playing field: $[x, y, \theta]^T$ Your goal is to implement a localization engine in C++ that estimates the three-dimensional state of the robot at each time step.

### 2.2 Landmark and World Description

The robot state is represented by a three-dimensional vector indicating the robot's position and orientation on the playing field: $[x, y, \theta]^T$ Your goal is to implement a localization engine in C++ that estimates the three-dimensional state of the robot at each time step. Each marker observation consists of the following three values:

- markerIndex - The index of the marker [0-3]
- distance - The observed distance to the landmark (in meters)
- orientation - The observed bearing to the landmark (in radians)

Similarly, the program controller module provides robot movement updates to your localization engine (observed state differences $[\Delta x, \Delta y, \Delta\theta]^T$ from odometry in local robot coordinates). The global (x, y) position of the robot is expressed in meters, and the robot orientation $\theta$ in radians $[-\pi, \pi]$. The origin of the global coordinate frame is located at the center of the playing field, with $+x$ and $\theta = 0$ directed to the right, and $+y$ and $\theta = \pi/2$ directed upwards. The starting location of the robot, and the location of all four landmarks are given at the start of program execution. The dimensions of the playing field are also provided, as discussed in the next section. Figure 1 shows the field layout, an example robot start position, and the global coordinate frame for reference.

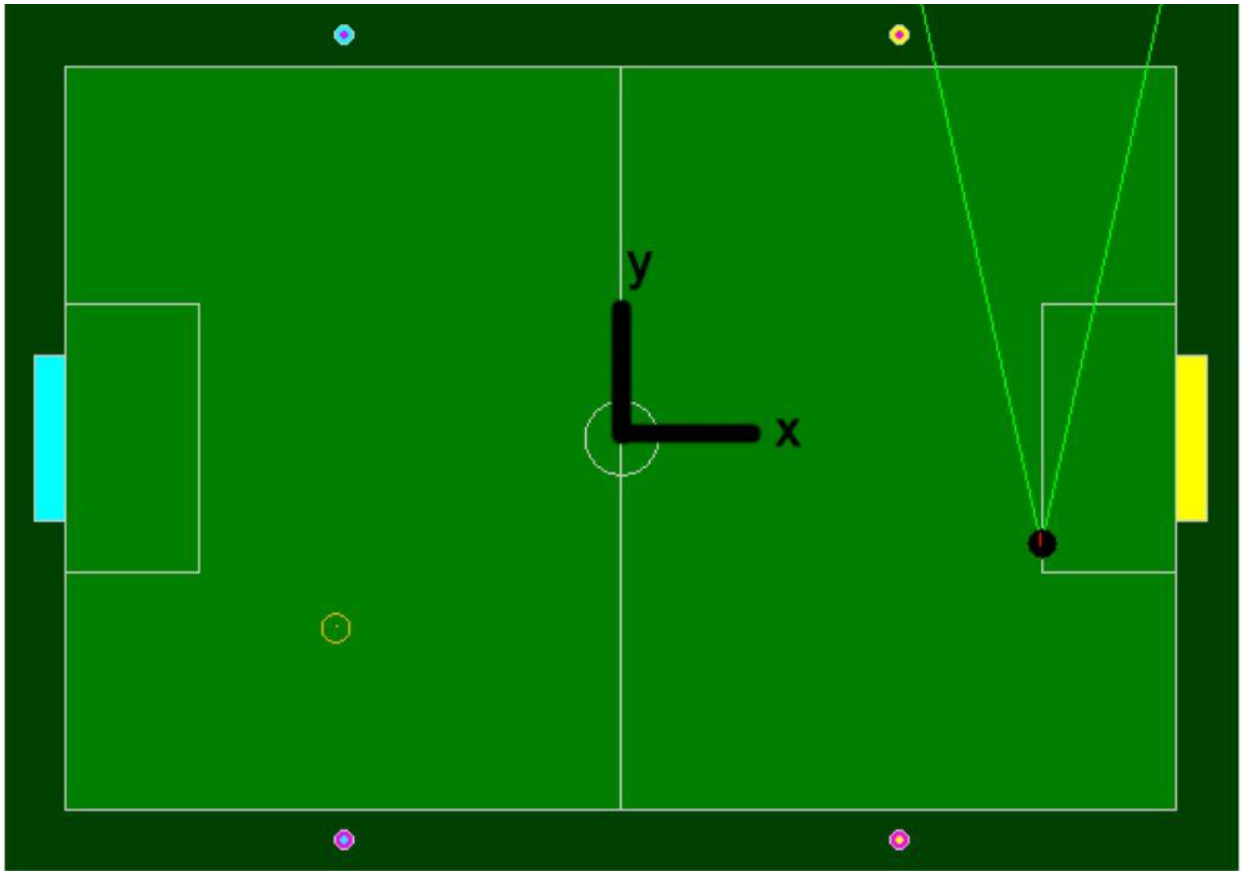

Figure 1: The field layout displaying the global coordinate frame, the four landmarks, an example robot start position (with sensor field of view shown), and an example goal position for the robot to achieve (orange circle).

**2.3 Robot Model**

The robot model/parameters are specified in a text file passed to the program at runtime, and include model parameters for both the odometric and sensor measurement noise. The robot motion model is differential drive. The robot parameters are:

- angle fov - The robot camera field of view angle (in degrees)
- sensor_noise_distance - The sensor noise in range measurement
- sensor_noise_orientation - The sensor noise in bearing measurement
- odom_noise_rotation_from_rotation - The noise in the odometric rotation estimate from robot rotational movement
- odom_noise_rotation_from_translation - The noise in the odometric rotation estimate from robot translational movement
- odom_noise_translation_from_translation - The noise in the odometric translation estimate from robot translational movement
- odom_noise_translation_from_rotation - The noise in the odometric translation estimate from robot rotational movement

## 3. Software

To help you get started quickly, we have provided all of the back-end software infrastructure needed to evaluate your localization engine. This includes all OpenGL window management, rendering of the field, goal location, robot actual position, and robot estimate position (as provided by your localization routines), and user mouse/keyboard input handling.Following is a description of the key software components:

- controller.a/.h - The static library containing the program controller. Responsible for all window management, user input handling, robot/goal rendering, pixel-to-global coordinate transformations, and triggering of localization engine routines (to be implemented by you).
- robot_defs.h - Contains definitions of important structures and constants, including field dimensions, pixel-to-meter ratios, robot state variables, and robot parameters.
- main.cpp/.h - The entry point of the program and container for all localization procedures, including sensor/motion updates, robot position estimate retrieval, and visualization of estimate uncertainty. The module contains (initially empty) function templates with space reserved for localization routines to be completed for the challenge.

## 4. Tasks

To successfully complete the localization challenge, please implement the following routines listed in main.cpp:

1. myinit() - Initialization routine for the localization engine. Takes as arguments the initial robot state, the robot parameters (read from input configuration file), and the locations of all four field markers.
2. motionUpdate() - Robot motion update step for localization engine.
3. sensorUpdate() - Sensor update step for localization engine.
4. getRobotPositionEstimate() - Sets the updated robot position estimate for the current time step.
5. mydisplay() - This function is called every time the display is updated. Draw some representation of the robot position uncertainty (e.g., particles, covariance ellipses, colorized grid-cells, etc.) and describe what method you chose in your write-up. You may use any OpenGL drawing routines.

Lastly, please create a README.md file describing your approach, implementation details, key

decisions, and anything you would like to improve if you had more time.

## 5. Evaluation

To test the localization engine, first compile the program and run it with a robot parameter file:

**make ./localization_test sample_input1.txt**

If the compilation fails, please make sure you have the required OpenGL/glut libraries installed on your system.

At startup, the initial robot position and goal position are chosen at random and the program is paused. To start the program, simply press the 'Enter' key. Once started, the program controller will command the robot to drive towards the goal until reaching the goal position, all the while calling the motionUpdate() and sensorUpdate() functions when necessary. When the robot reaches the goal, it will stop and wait for the next goal to be selected. To select a new goal position for the robot to pursue, simply left-click anywhere on the field and the robot will proceed to move toward the specified position. To introduce an unmodeled disturbance to the robot, simply right-click anywhere on the field to displace the robot to that location. Your implemented localization engine should be robust to these types of disturbances (known as the kidnapped robot problem).

To automate this procedure, press the Space bar. This will turn on 'Randomization' mode which will generate a sequence of goal positions at random, both with and without subjecting the robot to unmodeled disturbances. Here is a summary of all available user input commands:

- Enter key - Pause/unpause program
- Space bar - Enter/exit randomization mode (automated goal selection)
- 's' - Changes the camera scan mode [1-4] (front, sweep, cursor, fixed)
- 'x' - Disables all vision. Useful for testing odometry-only estimation.
- 'r' - Toggle display of robot/visual field of view on/off
- Left/right arrow keys - Changes the robot orientation manually
- Left mouse click - Selects new goal position at mouse location
- Right mouse click - Selects new robot position at mouse location

The program controller will take care of displaying your localization engine's current robot estimate at each time step (shown as a yellow circle) – in addition to printing the estimated position to the console window – but please remember to write your own draw routine to characterize the uncertainty in the estimate (and possibly adding new keyboard commands to toggle this display on/off).

To evaluate your localization engine, we will run a series of tests in different program modes and with different robot parameter files. Please make sure to test your localization engine under a variety of conditions, and especially under 'Randomization' mode.