SimMeR

Simulator for **Me**chatronics **R**obots

Ian Bennett

1 Abstract

This document presents the design and key features of a robotics simulator for MIE444 Mechatronics Principles. The simulator is a software tool written in MATLAB, designed to allow students to test navigation and localization algorithms remotely during the COVID-19 pandemic. The simulator provides software inputs and outputs for several different types of sensors and is plug and play compatible with a similar Bluetooth communication program used to send commands to a microcontroller on board a physical robot.

The program begins by generating a maze and user robot from a number of user editable text files. These files will contain the maze wall locations, ground checkerboard pattern, robot sensor suite and mounting locations, and control character mapping for the sensors and movement. Measurement and control precision is also definable by the user and can be configured to replicate the types of error seen in real world systems based on the configuration chosen by the user.

2 Introduction

The program is to be written in MATLAB using version 2020a, the most recent version available. No packages other than the standard install are to be used. An exception is made for open-source, freely available packages but only if they provide functionality that can't be easily replicated using the standard install packages.

The goal of the project component of the course MIE444 is to have students design and construct a robot to perform simple mapping and localization to autonomously collect a small object from within a maze and deposit into a predetermined zone.

To this end, students must program mapping and localization algorithms to take and fuse input data from a number of sensors and create output commands to control motors that will move the robot. The simulator will perform the following tasks as part of it's initial setup.

1. From a pre-defined text file, load the map including wall locations and checkerboard locations.
2. From a user-defined text file, generate a simulated robot including size and sensor loadout and placement.
3. Create a text file that data from each step the robot takes will be appended to for users to view and debug with.

As part of its main command loop, the simulator will follow the following steps:

1. Accept serial packet commands from a student designed algorithm, programmed in MATLAB or Python.
2. Funnel the packet to either the simulator or a bluetooth serial port connected to a physical robot
3. Parse the packet to extract the command character.
4. Search a user-defined lookup table to determine whether the packet is a telemetry request (i.e. to collect sensor data) or a control command to move the simulated robot in a direction, then act on the command.

a. For telemetry requests, pass these to a function that simulates the response data packet of the desired sensor. The function will take into account the position of the simulated robot within the maze, generate the value that the sensor should read, add noise based on a calibrated noise profile for that sensor, and construct a return data packet that is identical to one produced by the physical robot's microcontroller.

b. For control commands, these will be passed to a function that moves the simulated robot within the maze. The drive value and direction and/or the rotation value and direction for the configuration will be specified by the user algorithm, noise will be added based on a user-defined drive noise profile, and used to update the simulated robot. Several drive noise profiles will be pre-configured by the TAs based on common wheel/motor configurations.

5. For a control command, the program will generate a path based on the movement. If a collision is detected between the robot and the wall along the path, The program will assume that the robot stops moving at the collision point.

6. Based on the new location and/or rotation of the robot, positions of the body and sensors will be updated. For integration-based sensors (such as gyroscopes and odometers), their values will be updated based on the movement of the robot and noise functions

2.1 Compatibility

In order to ensure that the user-created programs work with both the simulator and a physical robot, it must take and interpret commands and respond to the environmental stimulus similarly. The input and output software interface should be the same. To this end, in addition to the simulator, a bluetooth command piping program will be created to pass data from the user program to a bluetooth module on the physical robot.

3 Configuration Files

This section details the various configuration files used by the robot simulator to do the following things:

1. Define the maze wall locations.
2. Define the maze checkerboard pattern.
3. Define the user's robot size, shape, sensor loadout, and sensor locations.
4. Define the user's robot's drive system and associated error/bias parameters.
5. Define the error parameters for each of the sensor types.
6. Define key mappings for each of the movement directions, rotation directions, and sensor telemetry requests.
    a. i.e. the string 'w-x' could correspond to moving forward (w) a number of inches (x), where x is a 16-bit float or similar. The string 'u1' would request data from ultrasonic sensor 1, whose position on the robot is defined by file #3.

This section will be filled in as the program is written.

4 Simulator Definition

This section details the functionality of the simulator and includes descriptions of the system-level functionality as well as descriptions of the functions that make it up.

4.1 State Diagram

The state diagram in Figure 1 shows the system-level process diagram for the robot simulator. Initialization tasks are in black, tasks related to the Bluetooth pipe are in blue, orange indicates tasks related to robot movement commands, and tasks related to querying sensors are in green (sorry for the messy diagram, I'll clean this up at some point). Red writing indicates a configuration file being queried.

4.2 Sensors

This section details the available sensors in the simulator, how each is designed to interact with the simulated environment, and how the error for each is determined.

4.2.1 Ultrasonic Rangefinder

4.2.2 Time of Flight Sensor

4.2.3 Line Following Sensor

4.2.4 Compass

4.2.5 Wheel Encoders

4.2.6 Gyroscope

4.3 Locomotion

The rest of this section will be filled in as the program is written.

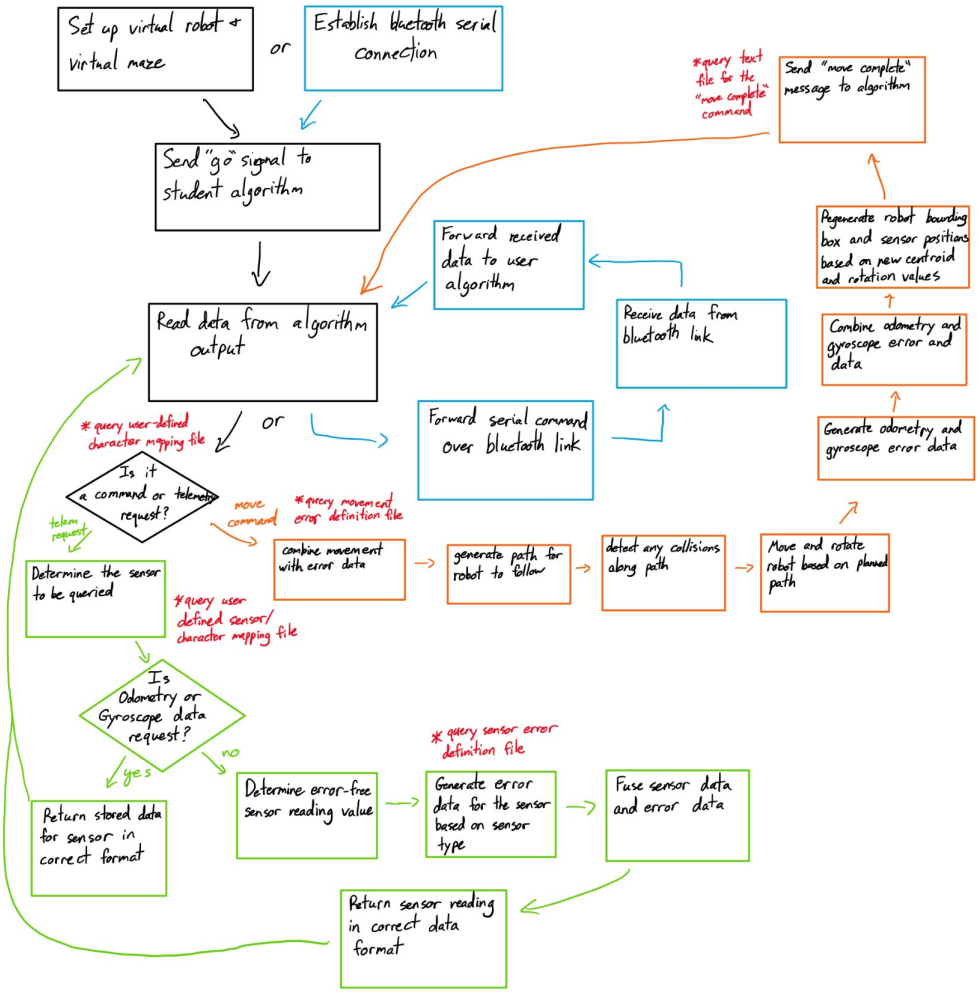

Figure 1: System level state diagram for the robot simulator.

5 Data Formats

This section details the data formats used for input to the simulator from the user's algorithm and output from the simulator to the user's algorithm. Also included is the expected input/output data format for the Bluetooth pipe program to/from the physical robot microcontroller.

This section will be filled in as the program is written.

6 Conclusion

This section will be filled in as the program is written.