IDP First Report – Software.

1. Specifications

Starting from the initial requirements, our team decided which functions had to be implemented in software. We also determined which interfaces the software would have to communicate over.

Functions to be implemented in software:
- Line following
- Junction identification
- 90 degree turns at junctions
- Position tracking
- Egg-processing mechanism control
- Egg-identification sensor data analysis

Interfaces:
- Launching and stopping of the software will be done through the wireless connection of the microcontroller to the departmental network.
- Readings from the electronics team's sensor boards will be input through the I2C port on the microcontroller
- Motors will be controlled directly from the microcontroller board, using the microcontroller's built-in PWM outputs

2. Preliminary experiments

In order to make informed high-level design choices, a number of preliminary experiments were undertaken to ascertain the capabilities of the various hardware interfaces. The results are presented here.

2.1. Latency and speed

Network latency was determined by sending a large number of test commands to the microcontroller sequentially, and measuring the average time taken per command if the program is run on a department workstation or directly on the microcontroller.

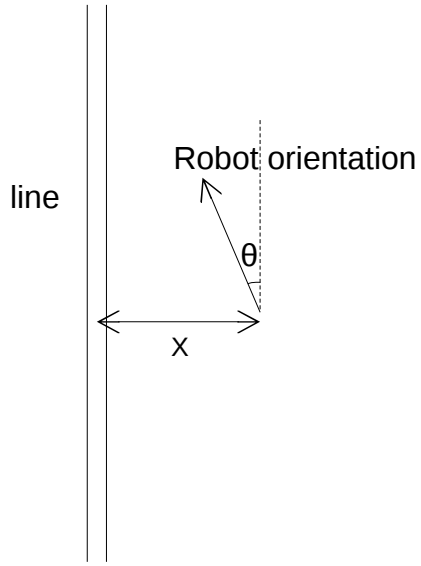
It was found that while the tests performed on the microcontroller consistently took 2ms each, those performed from the department workstation took between 4ms and 7ms, depending on network load.

On the other hand, when reading sensor information from the I2C bus every 10ms, it was found that the program was far more reliable at outputting information every 10ms when run on the workstation than on the microcontroller. This might be because of a different implementation of the delay function on the two machines. However, it is unlikely that this will be a crucial issue for the software, as we have precise time measurement via the stopwatch function.

From this, we can expect that although there are differences between running the software on a department workstation or on the microcontroller, the behaviour of the robot should not change substantially.

2.2. Line following and sensor position

Our first attempt at tackling this problem was by modeling the robot dynamics as a control system.



In the simple case where the robot is propulsed by two wheels, with speed respectively v_l and v_r, the kinematics of the robot are governed by the following equations:

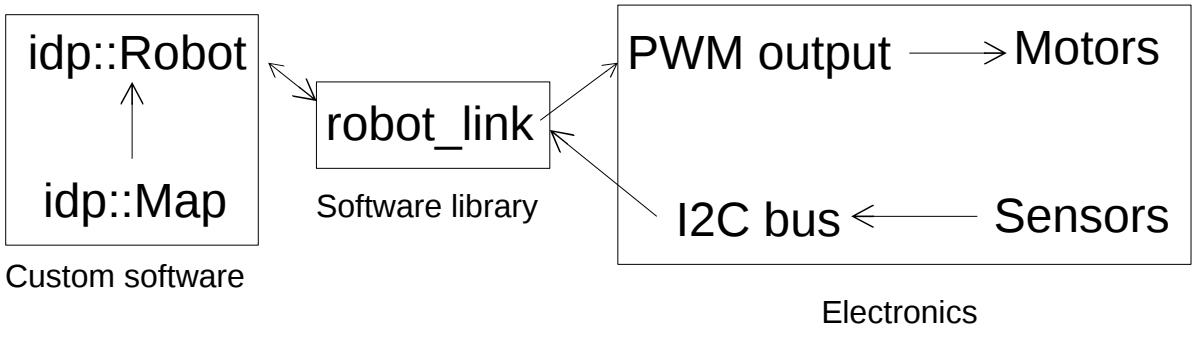
$$\frac{dx}{dt} = -v\sin\theta = -\frac{v_l + v_r}{2}\sin\theta \quad \text{and}$$

$$\frac{d\theta}{dt} = \omega_{robot} = \frac{-v_l + v_r}{2a} \quad \text{where a is the half-axle length.}$$

From this analysis, we elaborated our line following strategy (see paragraph 4.1).

3. High-level design

To fulfill the constraints identified in the specifications, and with the experience gained in the preliminary experiments, we came up with the following high-level design:



idp::Robot:

- Because the different functions outlined in the specification need to share a lot of information, it seems logical to store the robot state in a single class, on which member functions will perform operations.
- This will store not only low-level information such as current sensor values and the current time, but also higher-level, processed information such as the calculated trajectory of the robot.

- Member functions will operate on this class to provide functionality outlined in the specification.

idp::Map:

- This class contains a representation of the playing area, read from a configuration file.
- Allows the robot to be aware of points of interest in the playing area and have a higher-level awareness of position and trajectory.
- We chose to implement this in another class than idp::Robot, because it contains general map information, distinct from the robot state.

4. Initial Bill of Materials

Between 800 and 1200 lines of code in total, of which 440 are already written.

Source files:

- robot.h: main header file for the project, defines the main idp::Robot and idp::Map classes, as well as a few auxiliary classes.
- robot.cc: implementation of the functions operating on the idp::Robot class
- map.cc: implementation of the functions operating on the idp::Map class
- main.cc: “Duct tape” of the program: Implements the high-level logic, and puts together the functions defined in the above source files.

5. Next steps

We are already done implementing the link management and motor control functions in the idp::Robot class, as well as implementing the idp::Map class, totaling 440 lines of code.

The next step for us is to implement and test the line-following strategy, in order to have a working propulsion system. This is likely to take 4 days at least. Following that, we will focus on coding the egg-identification sensor analysis function, and the egg-processing mechanism function. Finally, we will concentrate on achieving higher-level awareness of the robot and system-testing.