# Teknologiprosjekt: Øving 2

**Planning report**

---

The project fits for industrial warehouses, hospitals and anywhere robots are suited to move things around automatically. There is a lack of cheap options in the current state of the art.
Our expected outcome is to make a robot follow a painted line, using a camera and detect objects in its path using an ultrasonic range finder. The robot should be able to move on any track and make decisions in real time. The overall goal is to create a cost-efficient line-following robot that can do simple tasks such as carrying cargo from A to B. The robot should be able work in different scenarios (harbor, warehouse, hospitals etc).

**Requirements**
There are a few requirements and tasks that we will follow in order to get this robot to do what we want, here is a specified list over some of them:

- Following any track with specific requirements. We were instructed to make our robot do the following:
  - Keep itself on the right hand lane, as if it was a car on a real road.
  - Turn right on every intersection or turn. (We are planning on implementing the ability to do more than just turning right, but we will keep this as the default as that is what we were instructed to do.)
- Regulate angular and linear velocity.
  - It is important for the robot to be able to adjust its velocity in order to not overshoot turns.
- Detect obstacles and respond to them.
  - Configuring the visual sensor to detect obstacles within a certain range and slowing down to a halt in order to not crash into anything.
- Carry small items as cargo.
  - The robot has a flat surface with a few screws sticking up, which are ideal to carry smaller objects.
- Control and get values / status in a web interface.
  - We will use a website as an interface, and implement remote controls to be able to take over and "drive" our robot. This interface will also contain values such as speed, angles and a live feed of what the robot is seeing through the cameras.

**Limitations**
As with any project there are limitations, either things we don't have the ability to do or lack of time and resources. Here are a few we stumbled across:

- Obstacle detection.

  - We are unable to make the robot move around obstacles, it can only stop and wait for the object to be removed.

- Cargo weight.
  - The cargo can't be too heavy as the robot lacks the power to carry heavier items.
  - The robot also lacks the ability to adjust speed according to the weight of the items carried, so the robot might lose cargo by going too fast.

- Line detection.
  - The track can only be made of a single colour for contrast comparison.

**Motion control algorithm**
In order to make the robot do its job, we need a motion control algorithm that at least loosely follows the points under:

1. Sample input data from the distance sensor and stop the robot if the distance is within a certain value.
   - We will install a distance sensor on the front end of the robot, we will then be able to measure how far the robot currently is from an object that is in its path. The robot will then be able to stop when he is within x mm of the object.
2. Check data from the image processing if the track exists in the image.
   - On track: Goto point 3.
   - Not on track: Drive around in circles until the track is found.
3. Check if the robot is near a crossroad.
   - On crossroad: Move either left. right or forward. The robot can be hardcoded to always move in one direction or take a random direction.
   - Not on crossroad: Go to point 4.
4. Find the shortest distance and angle to the middle of the track. This will be the general direction / the next destination.
5. Calculate angular and linear velocity according to the data sampled in point 3.
6. Check if it has reached it's final destination or goal. If that is not the case, go to point 1.

**I/O module**
This module contains code for every sensor on the robot, including output to the motors. This module will be responsible for connecting every other module together, it will be the core code of the robot.

**Camera module**
The project includes a Raspberry Pi camera. We will use the standard raspberry Pi camera library to receive images from the camera. Then we will use OpenCV to convert the image to an array of pixels.

**Distance sensor**
The project includes a distance sensor. The GoPiGo library contains a function that returns the distance to the obstacles in millimetres which we can use further in the code.

**Motor I/O**
A function to control the wheels of the robot. The function should take two parameters, one for the speed of the wheel and another that defines which wheel to set the speed.

**Image processing module**
In order for the robot to detect the road it's going to follow it will have to process images in real time, these images will be converted and scanned in order to make this as easy as possible:

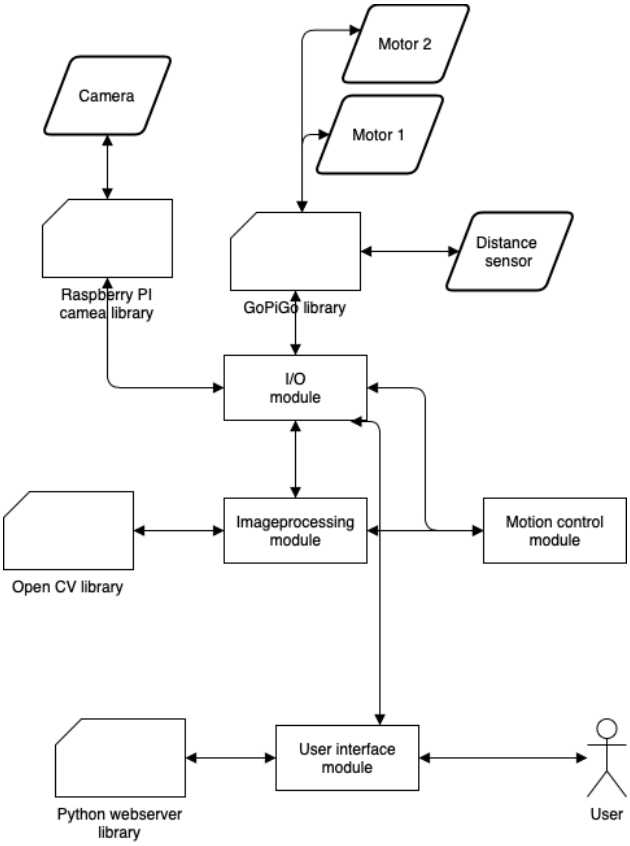
- Get the image as an array of pixels from the I/O module.
- Convert the colors in the array to black/white.
  - The robot is very limited when it comes to detecting colours, so to overcome that hurdle we will grayscale the image and use contrasting to make the image into black and white in order to detect the opposing colours.
- Use the Hough transform algorithm to find the edges of the track.
  - Using the Hough transform algorithm we can find the edges of the track, with these we can create a sort of coordination system that will tell the robot where to go.
- Find the middle of the road.
  - When we have the edges of the track and the coordination system it is easy to find the middle of the lane and stick to it.
- Return the middle of the road and the robots offset to the middle.

**User interface**
The user interface is a web page that is locally hosted on the raspberry pi. The webserver is running within the python program. The web page is using API calls to collect data / give command to the python program. All of this is happening asynchronous with the rest of the python code.

**Installation and deployment of the solution (Implementation)** In order to implement and deploy our solution we will follow these steps:

1. Installing the raspbian OS on a microSD card.
2. Setting up the raspberry pi to host a WIFI network.
3. Creating users for each group member on the linux system.
4. Setting up SSH.
5. Installing the GoPiGo libraries from github.
6. Opening port 80 on the firewall for the web interface.
7. Setting up a service in systemctl to run the python code when linux is booting.

Here is a visual representation of how all the different modules connects:



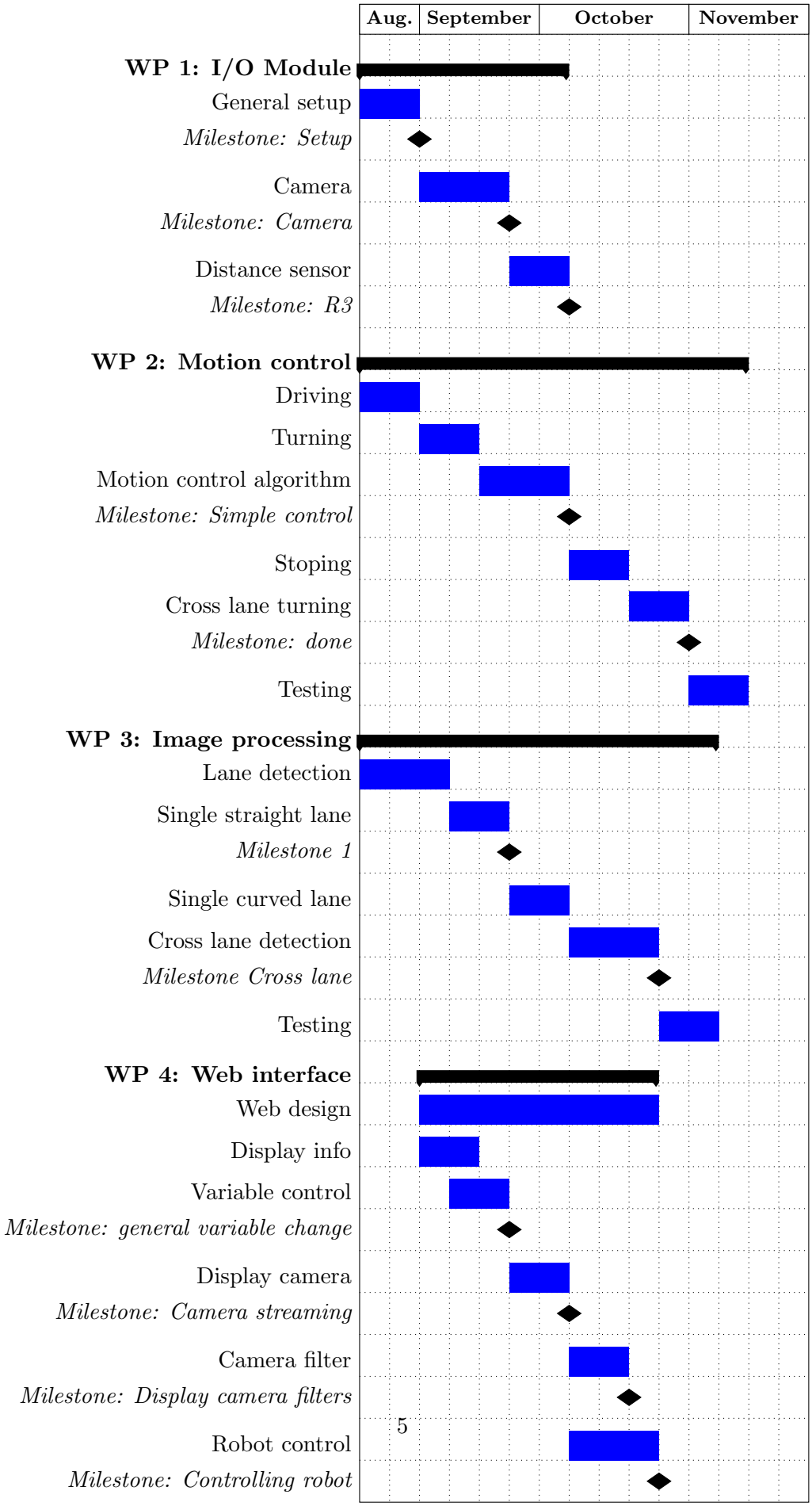

**Responsibilities**

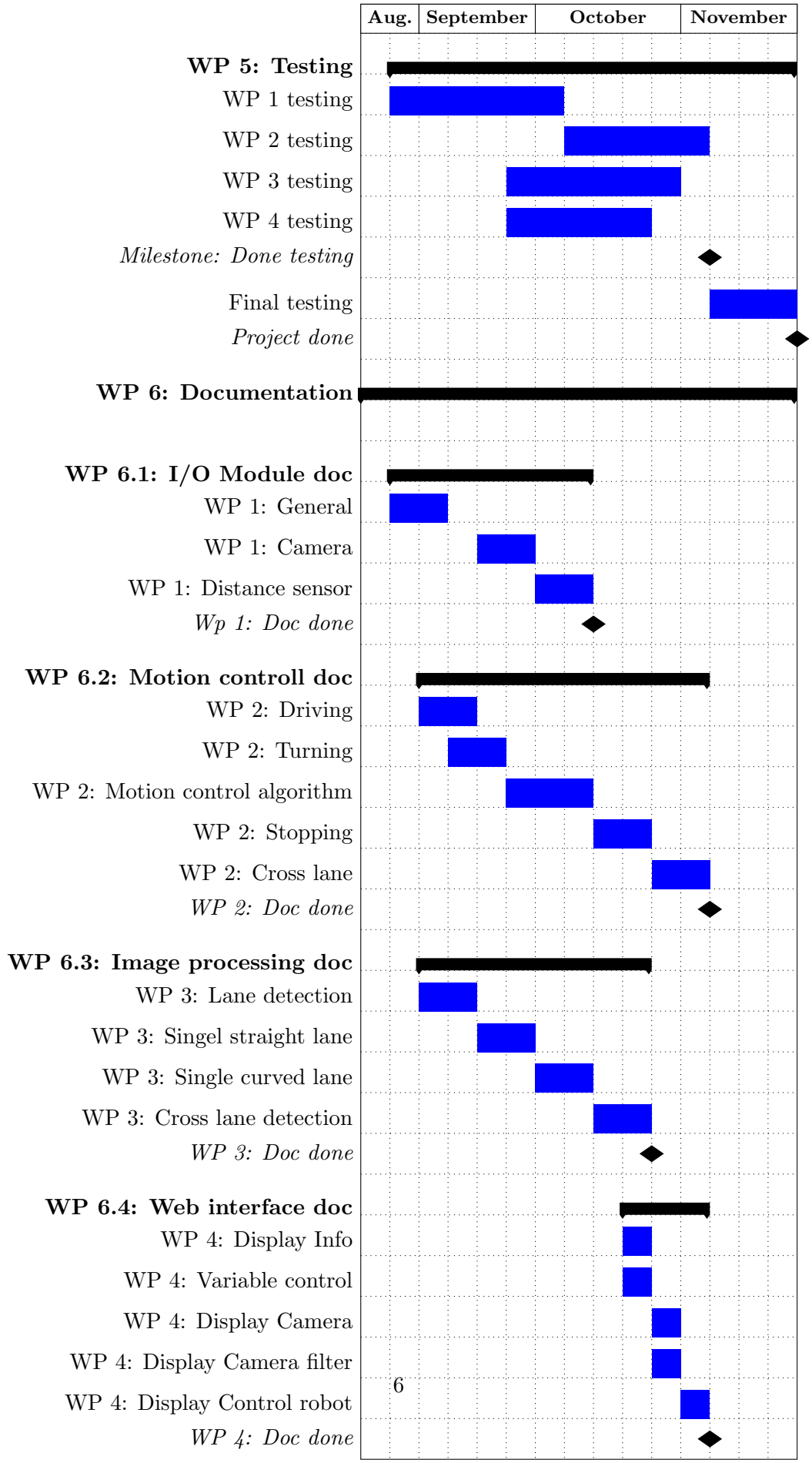
This is a list of the different responsibilities of the work packages. Everyone will do work in the different work packages, this list is only showing who is in charge of managing and leading the current work package.

| **Work package (WP)** | **Responsibility** |
|---|---|
| I/O module | Mathias |
| Motion control module | Tobias |
| Image processing module | Henrik |
| Web interface | Ruben |
| Testing | Andreas |
| Documentation | Erik |

**Gantt chart**

This Gantt chart is visualizing the planned workflow of the project.


Aug.
September
October
November
WP 1: I/O Module
General setup
Milestone: Setup
Camera
Milestone: Camera
Distance sensor
Milestone: R3
WP 2: Motion control
Driving
Turning
Motion control algorithm
Milestone: Simple control
Stoping
Cross lane turning
Milestone: done
Testing
WP 3: Image processing
Lane detection
Single straight lane
Milestone 1
Single curved lane
Cross lane detection
Milestone Cross lane
Testing
WP 4: Web interface
Web design
Display info
Variable control
Milestone: general variable change
Display camera
Milestone: Camera streaming
Camera filter
Milestone: Display camera filters
Robot control
Milestone: Controlling robot


Aug.
September
October
November
WP 5: Testing
WP 1 testing
WP 2 testing
WP 3 testing
WP 4 testing
Milestone: Done testing
Final testing
Project done
WP 6: Documentation
WP 6.1: I/O Module doc
WP 1: General
WP 1: Camera
WP 1: Distance sensor
Wp 1: Doc done
WP 6.2: Motion controll doc
WP 2: Driving
WP 2: Turning
WP 2: Motion control algorithm
WP 2: Stopping
WP 2: Cross lane
WP 2: Doc done
WP 6.3: Image processing doc
WP 3: Lane detection
WP 3: Singel straight lane
WP 3: Single curved lane
WP 3: Cross lane detection
WP 3: Doc done
WP 6.4: Web interface doc
WP 4: Display Info
WP 4: Variable control
WP 4: Display Camera
WP 4: Display Camera filter
WP 4: Display Control robot
WP 4: Doc done

**Risks**
There are always risks when implementing projects like this, some of the risks we can experience are:

- Camera lacking the resolution for our image processing algorithm.
  - One of the problems we might face is that the camera we are using will not work with our image processing algorithm. Some of the reasons it might not work is resolution or other specifications limitations on the camera we are using.
- Defect components (camera, distance sensor, etc.)
  - A risk when making a project that has multiple components is of course that some of the components we are using are broken in some way, this can be either easy to see physical damage and it can be hard to find internal damage.
- Bad placement of critical components (camera, distance sensor, etc.)
  - if the camera angle is either too high or too low the image processing could give bad results.
- Code that doesn't work the way we intended.
  - This could include everything from bugs and bad implementation to too high/low values.
- Bad time management.
  - When working on a larger project we run the risk of giving each task too much or too little time.

**Mitigation**
Some of the ways we can hopefully mitigate these risks are:

- Make sure that the components we use all have good enough specifications while still being affordable.
  - We can reduce the risk of limitations from components by carefully planning which components we need to use, finding out what specifications these components need to have, and making sure that we atleast fulfill the minimum requirements.
- Make sure that all components are working by thoroughly testing them before deployment.
  - This includes doing a physical inspection of them to try to see any broken parts, as well as thoroughly testing them before deployment.
- Thoroughly testing the code at each stage, to catch errors early and squash bugs.
- We can hopefully mitigate bad time management by thoroughly discussing and planning the project.