DeltaK - WRO 2025

DeltaK's Project Documentation - Kosain Abro, Rumi Kabir Ali, Rayyan Diwan

This document provides insight into the making of the DeltaK bot. We have a GitHub repo containing all files used and related to this project.

Understanding of the objective

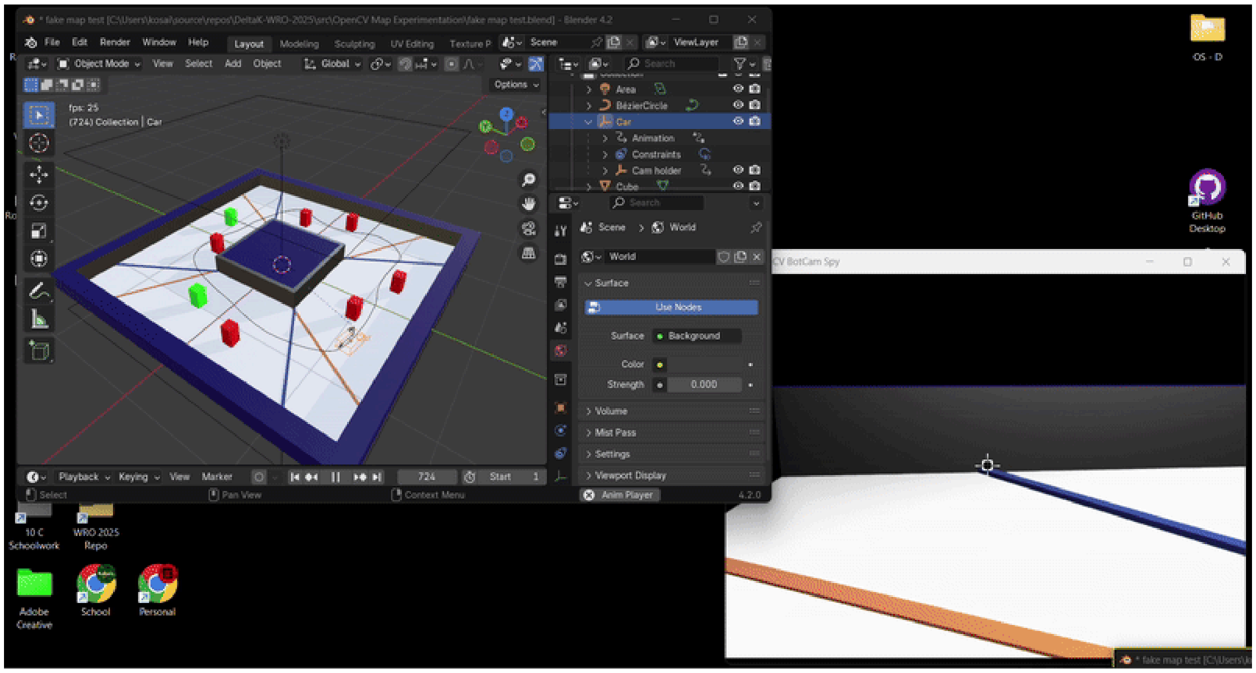
- **Restriction:** Shaft cannot be a standard L motor R motor.
- Dimensions: Max 300x200x300 mm (30x20x30cm) (XWidth, ZLength, YHeight)
- Max Width: 30cm
- Max Length: 20cm
- Max Height: 30cm
- Max Weight: 1.5 KG

Day 1

On our first day, we took the time to orient ourselves, setting the right direction. We had discussions mainly about the hardware first, focusing on what we would use. Different ideas were presented. One was using an RC car as a starting point. Another idea was to manufacture the parts in-house. We decided to go down the path of manufacturing our own, using a 3D printer for the chassis and frame.

Day 2

Two of the team members decided to work on finalizing the hardware list. While that took place, the third member decided to work a bit on getting familiar with OpenCV. So we initiated a basic animation on Blender that drives the car around on a rough 3D map recreation. We're feeding the camera input from the 3D camera mounted on the bot into a secondary window, which goes into OBS, which goes into a virtual camera that OpenCV may utilize. Below is a video demonstrating a basic program that uses OpenCV to highlight any pixel clusters of red or green:



Day 3

The Raspberry Pi, batteries, and camera module had arrived. We used an SD card to flash a fork of Linux on the Pi. In the meantime, one of us modeled an enclosure for the camera and printed it to fit a lens onto the module. A lens was very much needed, as we did some testing in the simulation. We found that an FOV of at least 90° was necessary for the virtual camera to see both walls. The FOV of the camera module, as listed on the website, is 53.50° with a bipolar deviation of 0.13°. Our target is greater than or equal to 90°. We also looked up its sensor size and focal length to perform calculations later on about its new FOV with the added focal length of our lens.

Day 4

We worked on scripting the APIs needed for driving the servo and motor drivers. We are using a servo driver to help with routing external power to the servo motor that controls the steering mechanism, whilst keeping data signals clean on the Raspberry Pi. We decided on a workflow regarding the RPI. Since it got pretty uncomfortable to program on the Pi itself fairly quickly, we repurposed our SD card as removable storage and moved the OS onto a faster

USB stick that is sleek and low profile, so as not to interfere with any components. This made it easier to code using our personal computers and then write the code to the SD card. We also implemented a few .sh files that run terminal commands to automatically load the Pi’s essential Python files and our programs with a double click on one file. This also made it easier to use the Pi in CLI mode, without the desktop environment taking up some performance. For the sake of testing hardware without the need for a monitor and keyboard, a simple temporary solution to that was running a Discord bot on the Pi. Using some of Discord’s basic message interaction features, we’re able to send signals to the motors wirelessly without the need to build a UI and interface with wireless functionality. This made it incredibly useful as a quick and easy way to test hardware with any changes made to the wiring.

Day 5

We had progress done on the actual hardware of the robot. We printed a redesigned frame for the entire bot and transferred components from the old frame to the new frame. We also had to use different components for both motors, which meant we would have to remake our API’s native output tools. The API used to deal with a brushless DC motor with an ESC (Electronic Speed Controller) which is tripolar compared to a regular DC motor with 2 terminals only. An ESC needs finicky calibration signals on powerup every single time, and the calibration signals that it needed were strangely not predictable at all. So we abandoned the ESC and switched to a regular DC motor that only takes in a regular DC voltage in forward or backwards polarity. The back DC motor is not complex at all so it takes up a small portion of our API’s code, since all it needs is some control in going forward and backward, as well as using PWM modulation to control the speed. We also redid the bracket that holds the motor.

Day 6

Today we did a little bit of both. We’re starting to connect the OpenCV software that was tested with the virtual map to the hardware-level functions that were written in C++. Since hardware outputs were tested on an Arduino, but our selected controller for the robot is a Raspberry Pi, we had to dedicate a day to port all logic from C++ to Python.