15th June - Planning Day:

# Everyone

* Set up GitHub (by the end of the first meeting)
* Remember git **pull**, BEFORE commit and push. It saves a lot of trouble.
* On GitHub, read documentations/ DRC\_Rules\_2024.pdf
* Run the relevant examples under /example\_code folder
  + You may need to pip install some libraries
  + Read the README.md in the home directory of the Git repository.

# Schedule (rough guide)

|  |  |
| --- | --- |
| Week 0  (10-16 June) | Starting on the GUI, mechanical design, colour mask and firmware. |
| Week 1  (17-23 June) | Mechanical + 3D printing. (Bryce)  Perspective transform. (Bryce)  Wireless connection + GUI. (Ib)  Colour masking: yellow, blue and purple. (JL)  Firmware for the car. (JZ)  Designing path planner (JL)  *Starting* to integrate UART thread into GUI (JZ).  By Saturday morning:   1. the GUI needs to show the **graph for perspective transform** with arrow indicating the path planner’s decision, since Bryce needs it. |
| Week 2  (24-30 June) | Before Monday:   1. **mechanical** needs to be finished. 2. perspective needs to be roughly tested **with the GUI** to make sure M matrix was generated correctly. Accuracy of +/- 10cm is fine. 3. **Playback\_gui.py** implemented (spec is below)   On Monday’s meeting:   1. Build a test track in a 7X7 metre area. 2. Integrate everything. 3. **Record videos** of car’s camera view while manually remote controlling the car, so **mechanical** needs to be complete. 4. Test JL’s path planner, if it is ready.   Rest of the week:   1. Design path planners (JL, JZ and Bryce).    1. Have a couple of different designs, so we can pick the best one later    2. Test/simulate using the **playback\_gui.py** and the **videos** recorded on Monday 2. **Integrate UART and scaffolding** for path planner into base\_gui.py. (JZ) 3. Dashcam\_gui.py (JZ)    1. A copy of the base\_gui.py but save the raw video feed automatically. Name of the video is the time stamp. This handy for investigating crashes in the future. |
| Week 3  (1-7 July) | 1. Path planner 2. Do the U-bend 3. Complete a lap without obstacles 4. Obstacle avoidance 5. Get something to act like the purple obstacles 6. Record a dashcam video of the car driving with obstacles on the track 7. Design an obstacle avoidance algorithm for path\_planner\_2.py 8. HSE and form |
| Week 4  (8-10 July) | 1. Calibrate colour masking to match QUT’s environment (JL) 2. Test perspective transform (Bryce) 3. Tune the firmware and planner, by finding the minimum PWM for moving the car. (JZ) 4. Fine-tune path planner (everyone) 5. Map speed to turning angle 6. Moving average for increasing the speed 7. Make it more biased to follow the blue 8. Prep a short presentation (should be pretty easy if our car works and can sell itself). 9. Win the competition. |

# Tasks

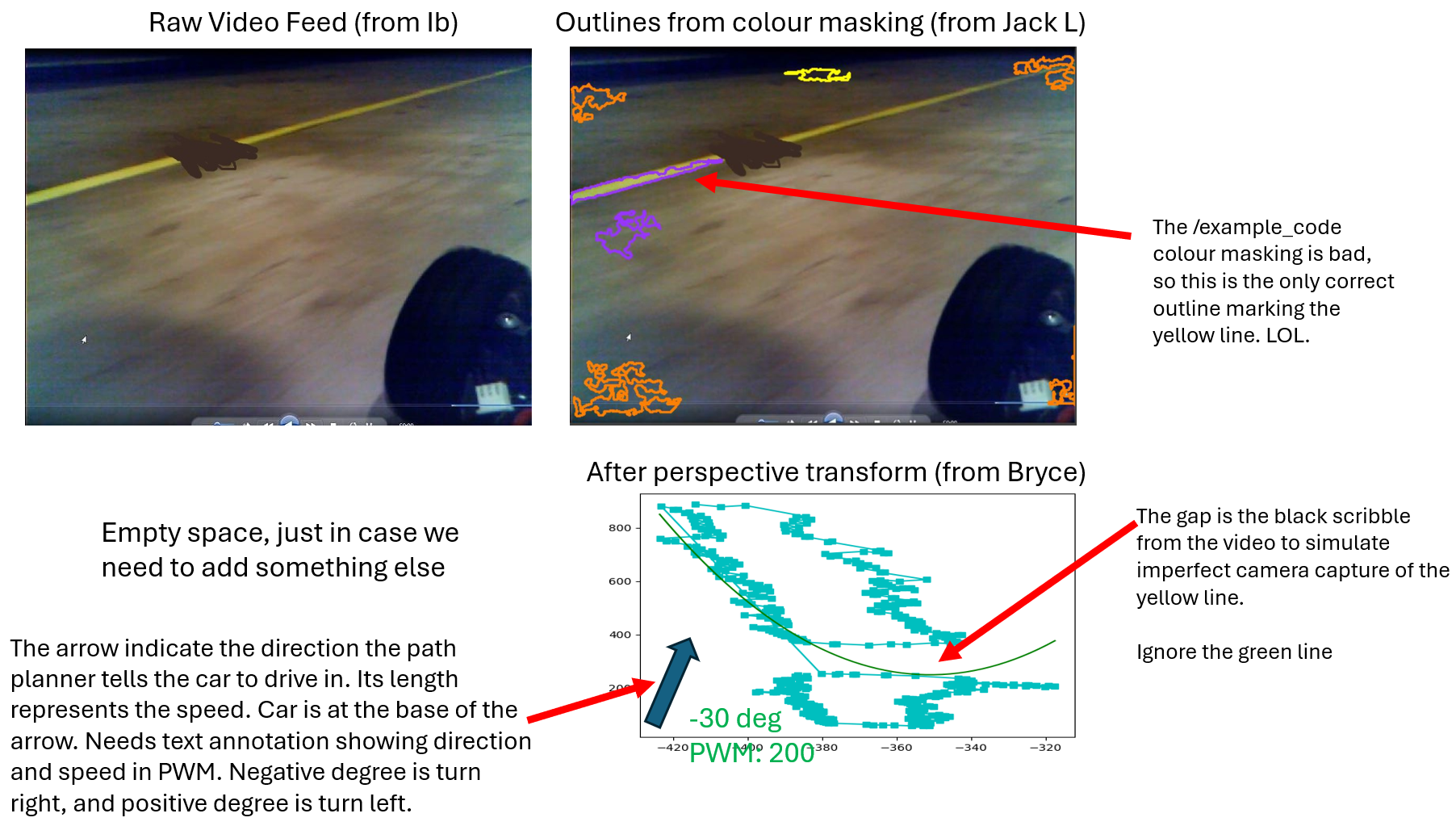
## Wireless video feed and GUI

Maybe use this app and example code:

<https://www.youtube.com/watch?v=0hT2cGSqPfk&ab_channel=Iknowpython>

GUI proposed layout:

|  |  |
| --- | --- |
| Raw video feed | Annotated video feed |
|  | Bird’s eye video of the track  The desired direction and speed of the car |



* The image above is just a very rough guide
* To get a **very rough** idea of how colour masking and perspective transforms are done, see ex\_colour\_mask.py and ex\_perspective\_transform.py under the example\_code folder.
  + Remember we care about **three** colours: purple (for obstacles), yellow and blue (which marks the side of the track). See example\_code/what\_track\_n\_obstacles\_look\_like.mp4
* For perspective transform:
  + X and y axis need to have the same scale
  + This scale is consistent across all frames of the video (a video is made of a sequence of frames of image)
  + The location of the car on the graph is fixed, so it does not jump around when the frame changes

# Playback GUI

* Mostly the same as base\_gui.py, but:
* The source video is a pre-recorded video instead of from IP Webcam.
* An interface to play/pause the video and select the timestamp to play like a YouTube video (the three parts circled below):

A green line on a black surface

Description automatically generated

# Colour Masking

* Test using the yellow and blue tapes.
* Pay attention to documentations/ DRC\_Rules\_2024.pdf Appendix A
* Three colours: yellow, blue and purple (the obstacles). Mask each colour individually, since different colours mean different things.
  + See example\_code/what\_track\_n\_obstacles\_look\_like.mp4
* Input: a frame from a raw video feed
* Output: the outline/edges around areas of interested colours (blue, yellow and purple).
* Manage glare and improve reliability (this was an issue last year)
* **May** need to consider other characteristics on top of colour, e.g. brightness, contrast, width, geometry.

# Perspective transform

* Depends on the camera placement, so this depends on mechanical task.
* Converts the camera point of view to a bird’s eye view (so it is easy to analyse the position of the edge of the track.
* Similar to homogenous/GPS frame transform from METR3100, but **different**.
* Input: the edges/outline/contour from the colour masking task, NOT the raw video feed. The edges will be expressed as array of x and y-coordinates in camera frame.
* Output: x and y-coordinates in bird’s eye view in centi-metre.
* The centre/origin of frame is at the centre of the phone’s camera.
* See the folder example\_code/from\_zac\_apelt, which has the instructions for perspective transform. Read the .jpg picture first. Zac did all of it last year, which is why I don’t know much about perspective transform.
  + To run the code in /from\_zac\_apelt, /from\_zac\_apelt needs to be **current** directory

# Mechanical and 3D printing

* Stability.
* Centre of gravity and reducing inertia, so it is easier for the car to turn.
* Camera placement: ensure a good field of view. (IMPORTANT)
  + Maybe hold the phone landscape???
  + Or tilt the phone???
* Phone:
  + Password: 123456
  + On the homescreen, IP webcam allows you to see the camera view remotely from a computer (might be handy). The instruction video (don’t worry about the Python part): <https://www.youtube.com/watch?v=0hT2cGSqPfk&ab_channel=Iknowpython>
* Bolts for securely attaching to the circuit boards (since gyroscope can’t wobble all the time)
* Needs to hold different configurations of power bank (since we need backup components)
* 3D print must be RED, since the majority of the car needs to be red (it is in the rules)
* Protection to damp the impact of a front-on collision. (Nice to have)
* Cable management (Nice to have)
* Preferable (**not necessary**) The corner of the board with resistor R3 and R4 is facing front right.

# Firmware

* Controls the motors on the car.
* reads from the gyroscope and control system.
* Receives the commands via Bluetooth using UART communication at 115200 baud rates.
* The car stops if UART/Bluetooth disconnected for a while.