B.Tech Project-1

Mid-Term Presentation

Vision Based Cruise Control with Obstacle Avoidance

Kartik Bansiwal (2021EE30743)
Shreysh Verma (2021EE30742)

Under the Supervision of
Prof. Indra Narayan Kar

Motivation

- Conduct applied research on control algorithms by developing an experimental setup from the ground up
- Gain deeper insight into the challenges and intricacies of control engineering in real-world applications
- The system will serve as a platform for future experimental research and testing in control engineering

Problem Statement

Practical Implementation of Adaptive Cruise Control Using 2-Wheeled Differential Drive Robots with Obstacle Avoidance

- Objective: Explore the application of standard control algorithms on a two-wheel differential drive robot
- Focus: Adaptive cruise control with obstacle avoidance
- Methodology: Implement and validate algorithms experimentally in a practical setup
- Outcome: Understand challenges and limitations of control systems in robots

Requirements

Hardware:

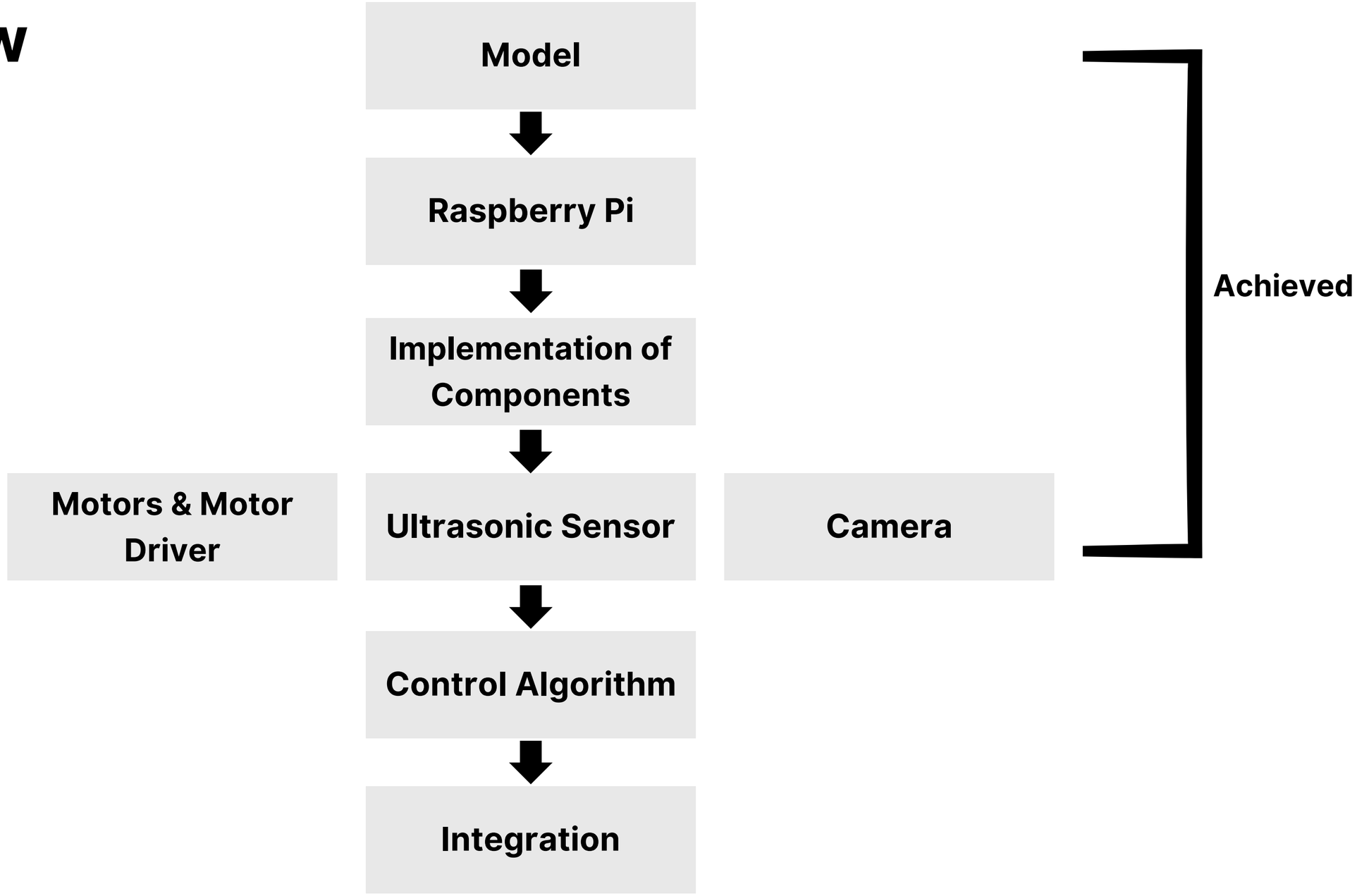
1. Two Wheeled Chassis with Castor Wheel x1
2. 12V DC Motors with Encoder x2
3. L298N Motor Driver x1
4. Raspberry Pi 4B+ x1
5. Camera Module (PiCam/ USB Cam) x1, Depth Camera x1 (As backup)
6. HC-SR04 Ultrasonic Sensor x1
7. Power Distribution Board (PDB) x1
8. 20000 mAh Power Bank x1
9. OLED Display x1
10. Wires & Pin Connectors

Software:

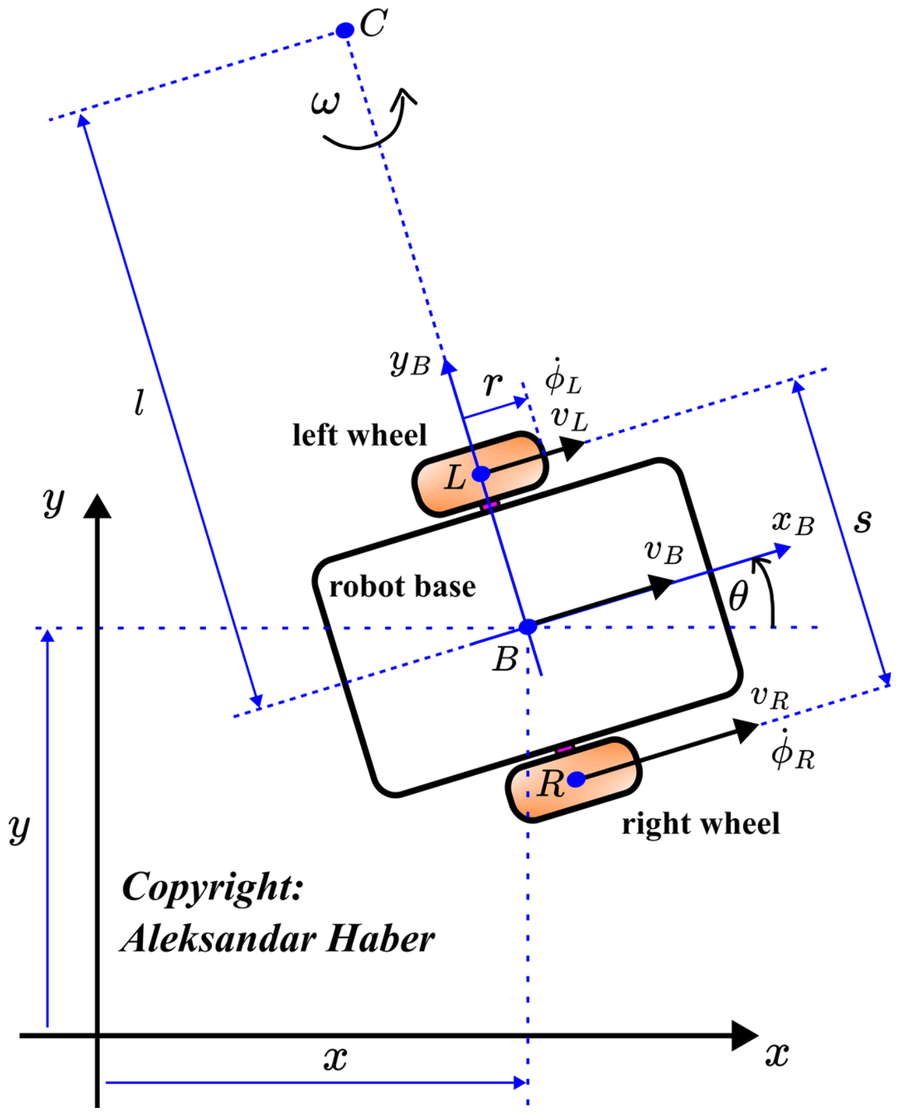
1. RaspberryOS
2. Matlab
3. Simulink
4. Jupyter

Draft List of Products: https://docs.google.com/document/d/1JFgS9_XhMZYiK7PhyH7aMhg7rbPVlil24OEdyPLiHJU/edit?usp=sharing

Workflow


Model
Raspberry Pi
Implementation of Components
Motors & Motor Driver
Ultrasonic Sensor
Camera
Control Algorithm
Integration
Achieved

Model Kinematics



$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r\cos(\theta)}{2} & \frac{r\cos(\theta)}{2} \\ \frac{r\sin(\theta)}{2} & \frac{r\sin(\theta)}{2} \\ -\frac{r}{s} & \frac{r}{s} \end{bmatrix} \begin{bmatrix} \dot{\phi}_L \\ \dot{\phi}_R \end{bmatrix}$$

$\dot{x}$ is the projection of the velocity v_B on the x axis.

$\dot{y}$ is the projection of the velocity v_B on the y axis.

r is the radius of the wheels.

s is the distance between the points L and R.

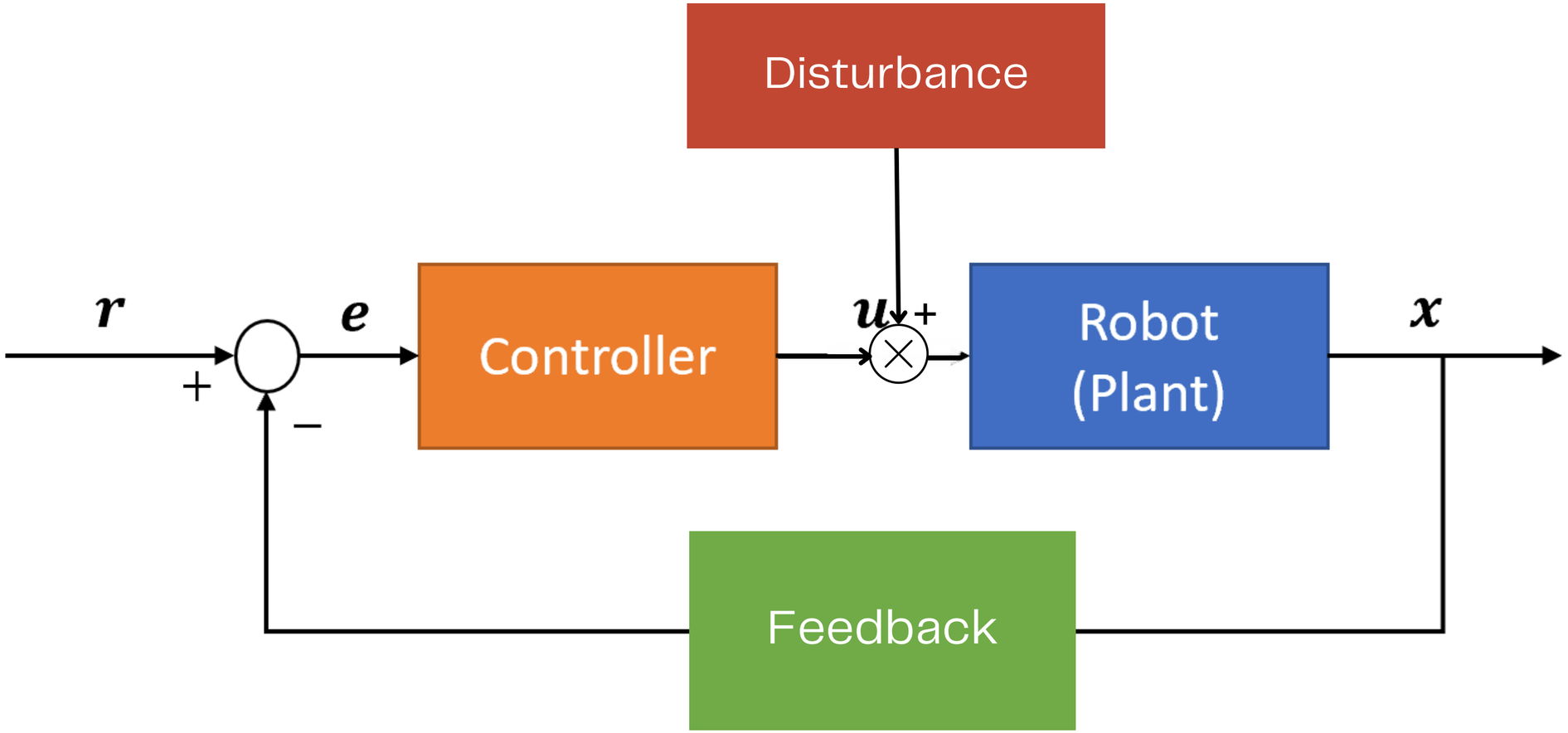
$\dot{\phi}_L$ is the angular velocity of the left wheel.

$\dot{\phi}_R$ is the angular velocity of the right wheel.

θ is the angle of rotation of the robot which is at the same time the angle between the body frame and the inertial frame

Ref: https://aleksandarhaber.com/clear-and-detailed-explanation-of-kinematics-equations-and-geometry-of-motion-of-differential-wheeled-robot-differential-drive-robot/

Block Diagram

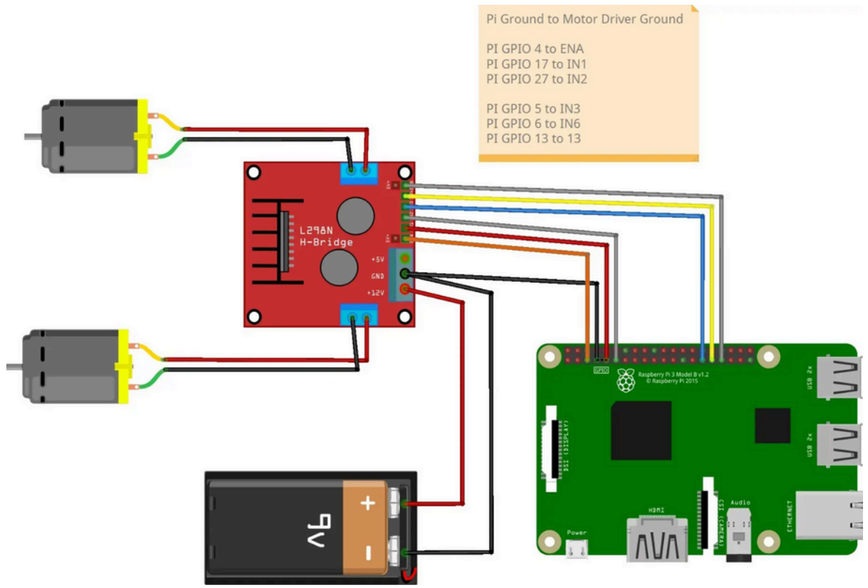


Ref: https://ucr-robotics.readthedocs.io/en/latest/tbot/moRbt.html#control-of-the-wheeled-robots

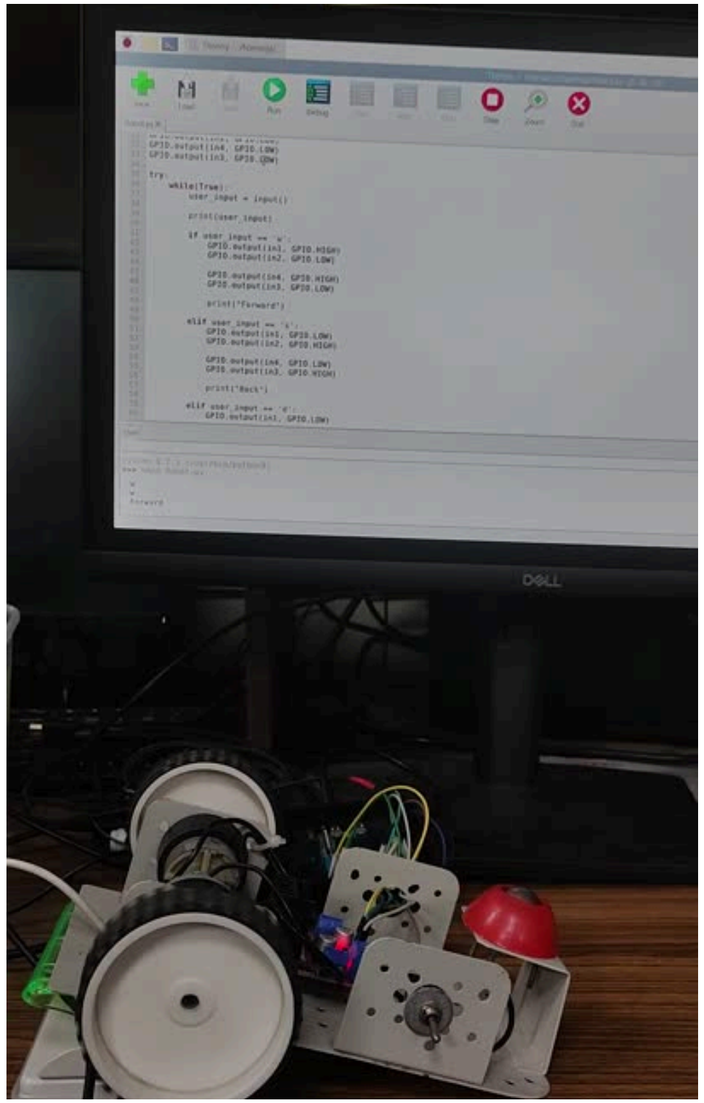
12V DC Motors with L298N Driver

Experimental Setup: Raspberry Pi 4B+, Chassis with two 12V DC motors controlled by L298N driver.
Code: Executed on Thonny IDE & Raspberry Pi 4B+, operated the motors at 75% speed in both forward and reverse directions.

The motors were driven by a **PWM** signal with a frequency of **100Hz** and a duty cycle of **75%**, resulting in **75%** of the motor's maximum speed.



Circuit Diagram



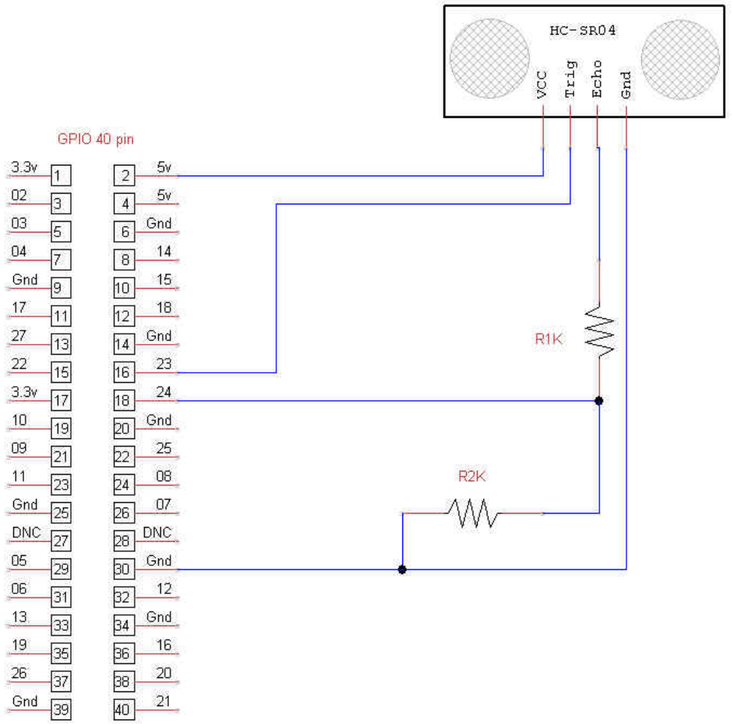
Demonstration + Code + Result

Ref: https://www.collvy.com/blog/controlling-dc-motor-with-raspberry-pi

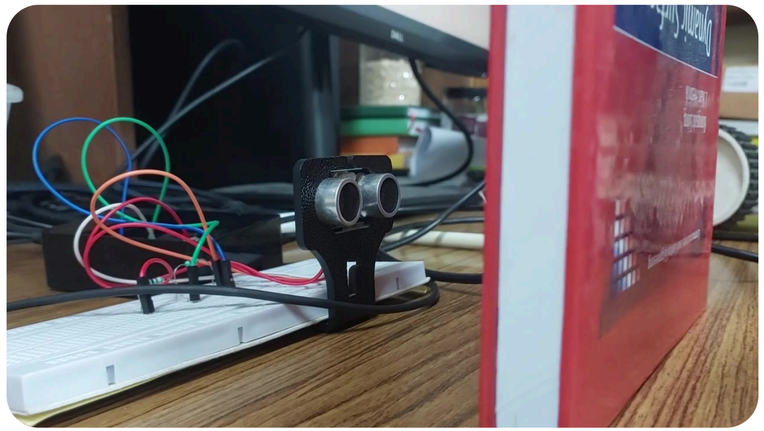
Ultrasonic Sensor

Experimental Setup: HC-SR04 Ultrasonic Sensor, 1k Ω resistor, 2k Ω resistor & Raspberry Pi 4B+

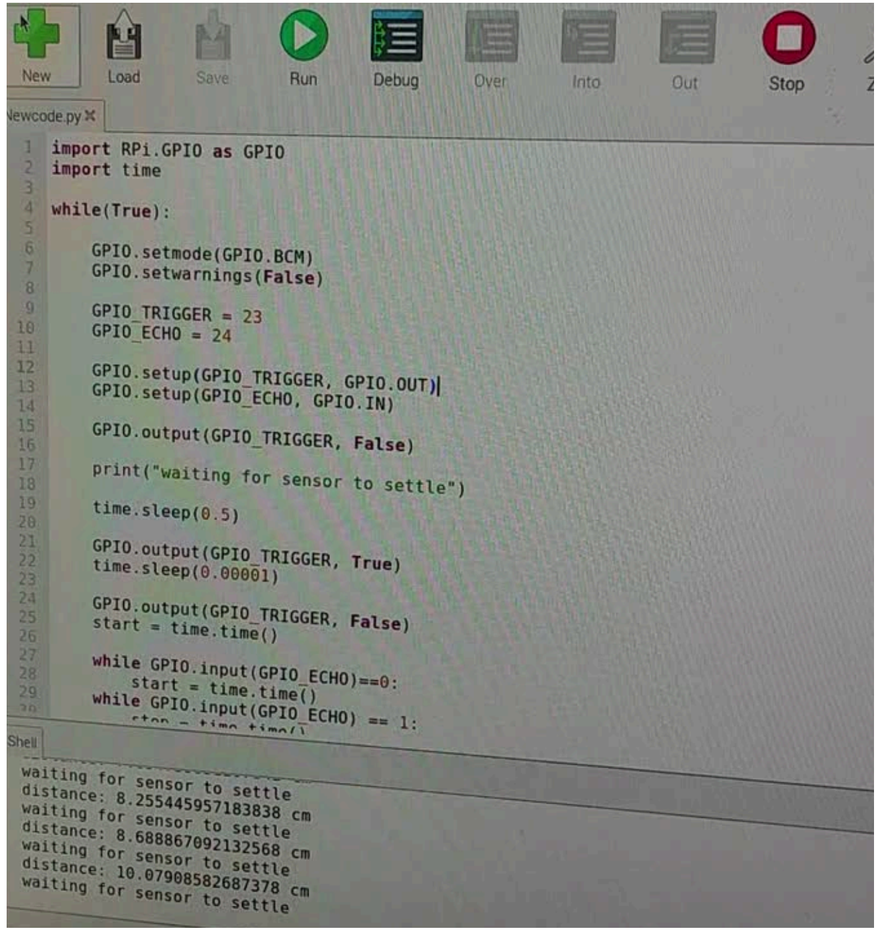
Accuracy: Accurate measurements within a margin of **+/- 1 cm**



Circuit Diagram



Demonstration



Code + Result

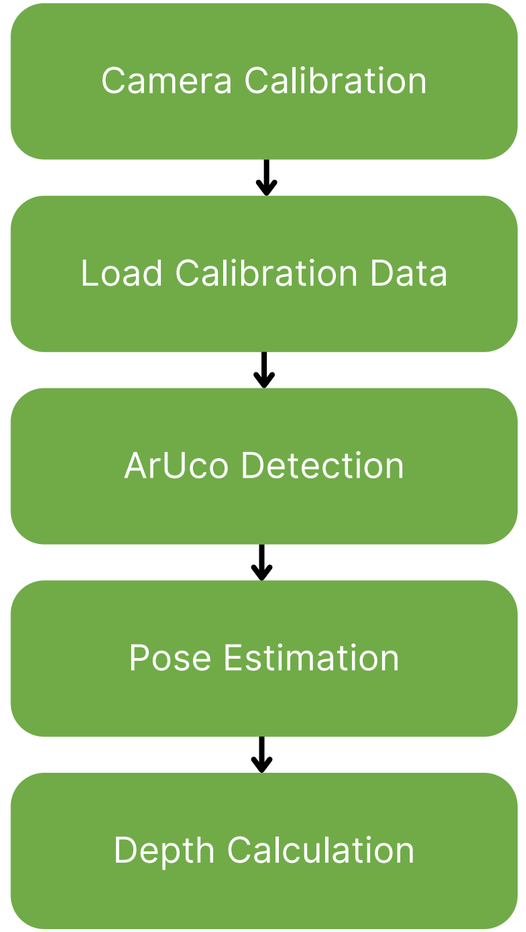
Ref: https://forums.raspberrypi.com/viewtopic.php?t=206751

Feedback from Camera:

ArUco Marker Detection and Real Time Depth Estimation

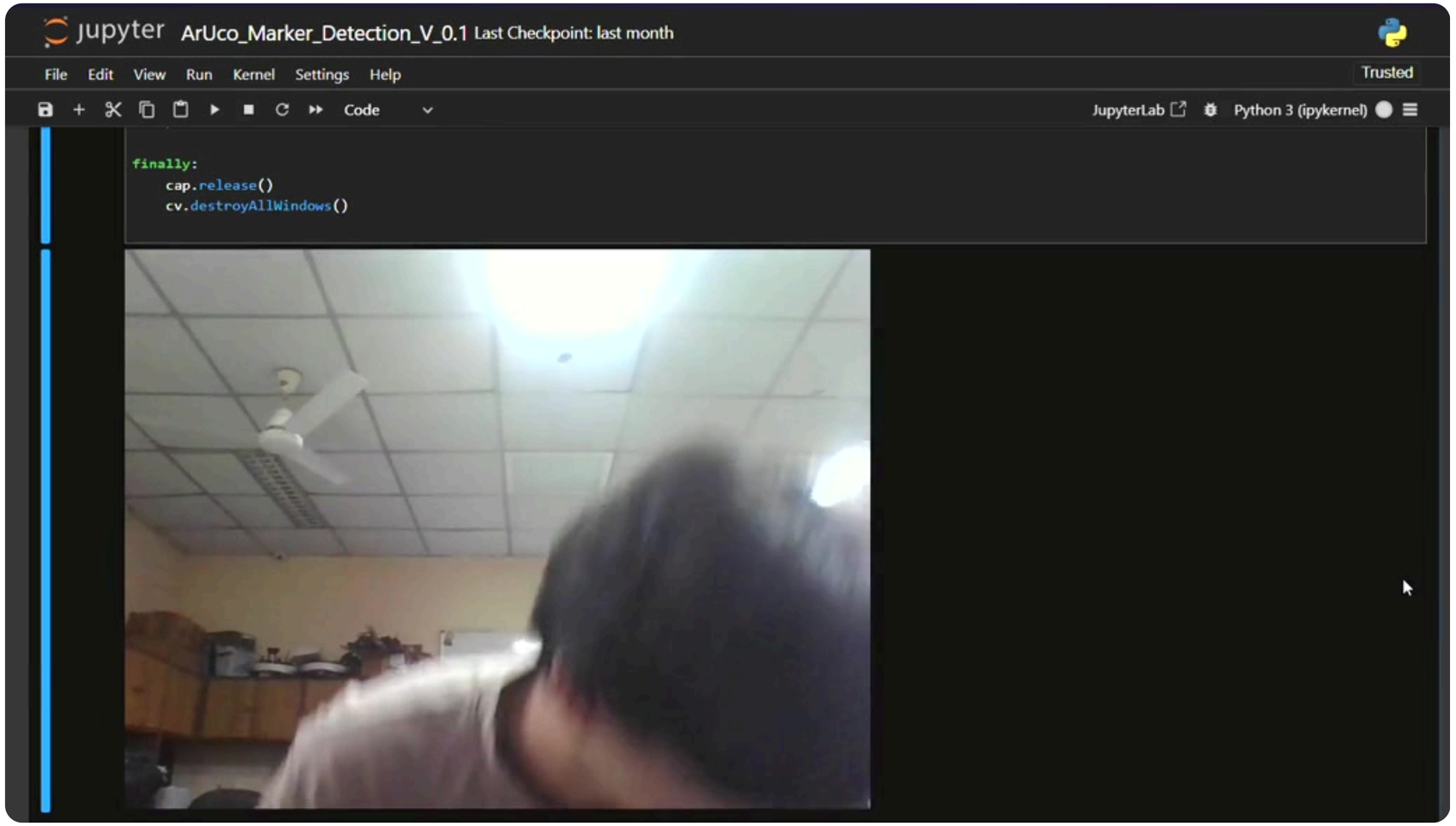
Initialization (Monocular Camera Callibaration):

- The camera was calibrated with over 150 images of a chessboard grid.
- The grid was positioned at various locations in space for calibration.
- This calibration accounted for radial and tangential distortions.
- It enabled the extraction of the camera's intrinsic and extrinsic properties.



Ref: https://www.youtube.com/watch?v=YOpJrB6bQxo

Implementation:



The implementation involved distance measurements of two ArUco markers, both individually and simultaneously, to evaluate the effectiveness of distance measurement and proper differentiation between the markers.

Accuracy: Accurate measurements within a margin of **+/-5 cm**

Ref: https://www.youtube.com/watch?v=YOpJrB6bQxo

Result

- Showcased the working of the components (12V DC Motors, L298N Driver, Ultrasonic Sensor) separately
- The system achieved object detection with a margin of error within **+/- 5 cm**, and ultrasonic sensors had a precision of **+/- 1 cm**

Next Tasks

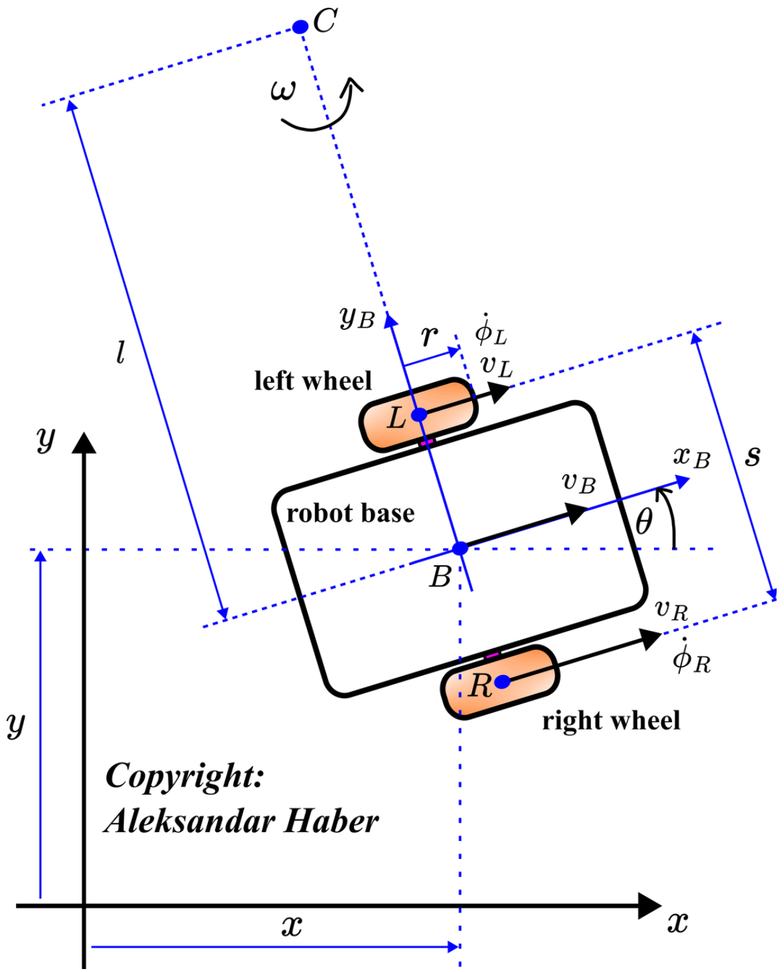
- Order Components: Finalize and order the hardware components needed
- Enhance Obstacle Detection Range: Combining visual and ultrasonic feedback for effective obstacle detection and avoidance
- Control Algorithm
- Integration
- Testing and Documentation

B.Tech Project-1

Mid-Term Presentation

Thank You!

Model Kinematics: Derivation



That is, we start from the assumption that the following quantities and parameters are known $\dot{\phi}_L$, $\dot{\phi}_R$, s, and r, and we want to determine $\dot{x}$, $\dot{y}$, and $\dot{\theta}$.

$$\begin{aligned} v_L &= \omega(l-\frac{s}{2}) \\ v_R &= \omega(l+\frac{s}{2}) \end{aligned} \longrightarrow \omega = \frac{v_L}{l-\frac{s}{2}} \longrightarrow \begin{aligned} v_R &= \frac{v_L}{l-\frac{s}{2}}(l+\frac{s}{2}) \\ v_R(l-\frac{s}{2}) &= v_L(l+\frac{s}{2}) \\ v_R l - v_R\frac{s}{2} &= v_L l + v_L\frac{s}{2} \\ l(v_R-v_L) &= v_R\frac{s}{2} + v_L\frac{s}{2} \end{aligned} \longrightarrow l = \frac{s(v_R+v_L)}{2(v_R-v_L)}$$

$$\begin{aligned} v_L &= \omega(l-\frac{s}{2}) \\ v_L &= \frac{\omega s}{2}\left(\frac{v_R+v_L}{v_R-v_L}-1\right) \\ v_L &= \frac{\omega s}{2}\left(\frac{2v_L}{v_R-v_L}\right) \end{aligned} \longrightarrow \omega = \frac{v_R-v_L}{s} \longrightarrow \begin{aligned} \dot{x} &= v_B\cos(\theta) \\ \dot{y} &= v_B\sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \longrightarrow \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\cos(\theta) & 0\\ \sin(\theta) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v_B\\ \omega\end{bmatrix}$$

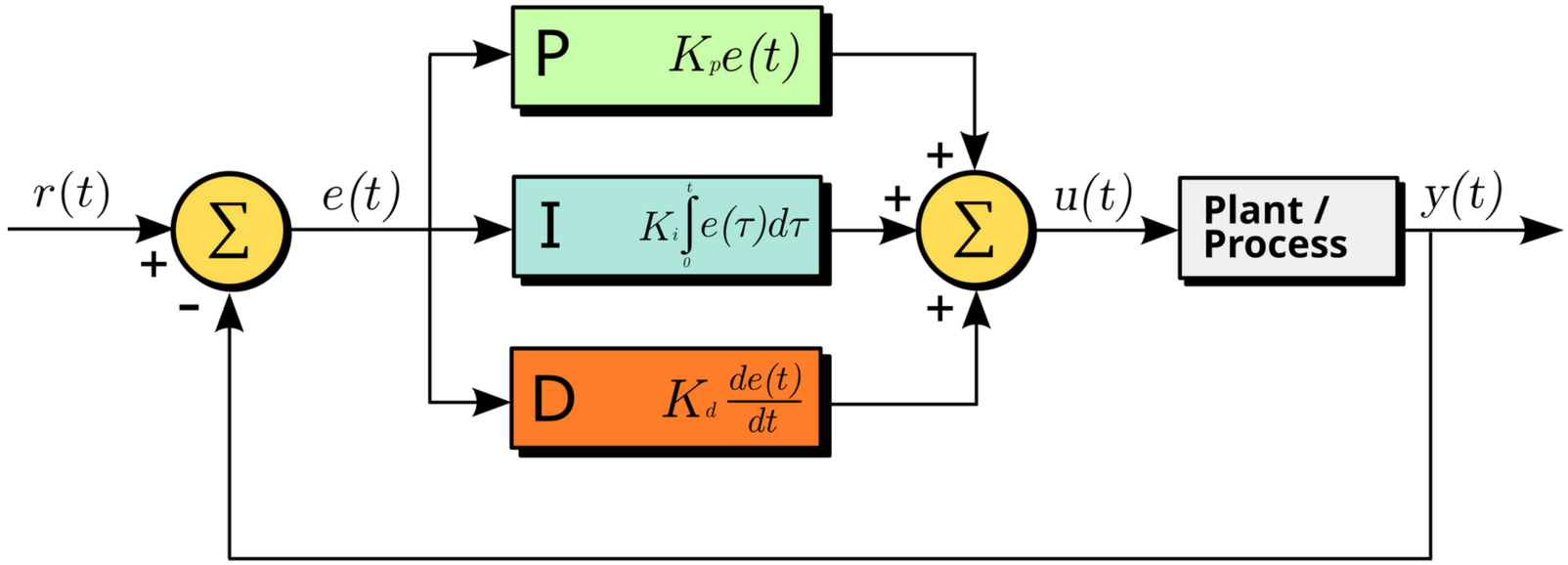
$$v_B = \omega\cdot l \longrightarrow \begin{aligned} v_B &= \frac{v_R-v_L}{s}\cdot\frac{s}{2}\frac{(v_R+v_L)}{(v_R-v_L)} \\ v_B &= \frac{v_R+v_L}{2} \end{aligned} \longrightarrow \begin{aligned} v_B &= \frac{v_R+v_L}{2} \\ \omega &= \frac{v_R-v_L}{s} \end{aligned} \longrightarrow \begin{bmatrix}v_B\\ \omega\end{bmatrix} = \begin{bmatrix}\frac{1}{2} & \frac{1}{2}\\ -\frac{1}{s} & \frac{1}{s}\end{bmatrix}\begin{bmatrix}v_L\\ v_R\end{bmatrix}$$

$$\begin{aligned} v_L &= r\dot{\phi}_L \\ v_R &= r\dot{\phi}_R \end{aligned} \longrightarrow \begin{bmatrix}v_L\\ v_R\end{bmatrix} = \begin{bmatrix}r & 0\\ 0 & r\end{bmatrix}\begin{bmatrix}\dot{\phi}_L\\ \dot{\phi}_R\end{bmatrix} \longrightarrow \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\frac{r\cos(\theta)}{2} & \frac{r\cos(\theta)}{2}\\ \frac{r\sin(\theta)}{2} & \frac{r\sin(\theta)}{2}\\ -\frac{r}{s} & \frac{r}{s}\end{bmatrix}\begin{bmatrix}\dot{\phi}_L\\ \dot{\phi}_R\end{bmatrix}$$

Ref: https://aleksandarhaber.com/clear-and-detailed-explanation-of-kinematics-equations-and-geometry-of-motion-of-differential-wheeled-robot-differential-drive-robot/

PID Controller

The controller attempts to minimize the error over time by adjustment of a control variable u(t)



A block diagram of a PID controller in a feedback loop. r(t) is the desired process variable (PV) or setpoint (SP), and y(t) is the measured PV.

Working of the System

1. Reference Setting: The system starts with the reference input (r), which could be the target position for the robot.
2. Error Calculation: The current position from the sensors are compared to the reference value at the summing junction to calculate the error (e).
3. PID Controller: The PID controller processes the error and generates a control signal (u). The control signal adjusts the motors of the differential drive robot to reduce the error.
4. Robot Execution: The control signal (e.g., speed commands to the motors) moves the robot.
5. Sensor Feedback: The sensors (including encoders and camera) provide real-time feedback on the robot's current state, allowing the system to continuously adjust.
6. Obstacle Avoidance: If an obstacle is detected by the camera module, the obstacle detection module adjusts the control signal to avoid collisions, possibly slowing down or rerouting the robot.

Feedback from Camera:

ArUco Marker Detection and Real Time Depth Estimation

Radial distortion can be represented as follows:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Similarly, tangential distortion occurs because the image-taking lense is not aligned perfectly parallel to the imaging plane. So, some areas in the image may look nearer than expected. The amount of tangential distortion can be represented as below:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

In short, we need to find five parameters, known as distortion coefficients given by:

$$Distortion\ coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

In addition to this, we need to some other information, like the intrinsic and extrinsic parameters of the camera. Intrinsic parameters are specific to a camera. They include information like focal length (f_x, f_y) and optical centers (c_x, c_y). The focal length and optical centers can be used to create a camera matrix, which can be used to remove distortion due to the lenses of a specific camera. The camera matrix is unique to a specific camera, so once calculated, it can be reused on other images taken by the same camera. It is expressed as a 3x3 matrix:

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Extrinsic parameters corresponds to rotation and translation vectors which translates a coordinates of a 3D point to a coordinate system.

For stereo applications, these distortions need to be corrected first. To find these parameters, we must provide some sample images of a well defined pattern (e.g. a chess board). We find some specific points of which we already know the relative positions (e.g. square corners in the chess board). We know the coordinates of these points in real world space and we know the coordinates in the image, so we can solve for the distortion coefficients. For better results, we need at least 10 test patterns.

As mentioned above, we need at least 10 test patterns for camera calibration. OpenCV comes with some images of a chess board (see samples/data/left01.jpg – left14.jpg), so we will utilize these. Consider an image of a chess board. The important input data needed for calibration of the camera is the set of 3D real world points and the corresponding 2D coordinates of these points in the image. 2D image points are OK which we can easily find from the image. (These image points are locations where two black squares touch each other in chess boards)

What about the 3D points from real world space? Those images are taken from a static camera and chess boards are placed at different locations and orientations. So we need to know (X, Y, Z) values. But for simplicity, we can say chess board was kept stationary at XY plane, (so Z=0 always) and camera was moved accordingly. This consideration helps us to find only X,Y values. Now for X,Y values, we can simply pass the points as (0,0), (1,0), (2,0), ... which denotes the location of points. In this case, the results we get will be in the scale of size of chess board square. But if we know the square size, (say 30 mm), we can pass the values as (0,0), (30,0), (60,0), ... Thus, we get the results in mm. (In this case, we don't know square size since we didn't take those images, so we pass in terms of square size).

3D points are called **object points** and 2D image points are called **image points.**