CHAPTER-2

System Design

2.1 Project Architecture

Our proposed system consists of the following four modules:

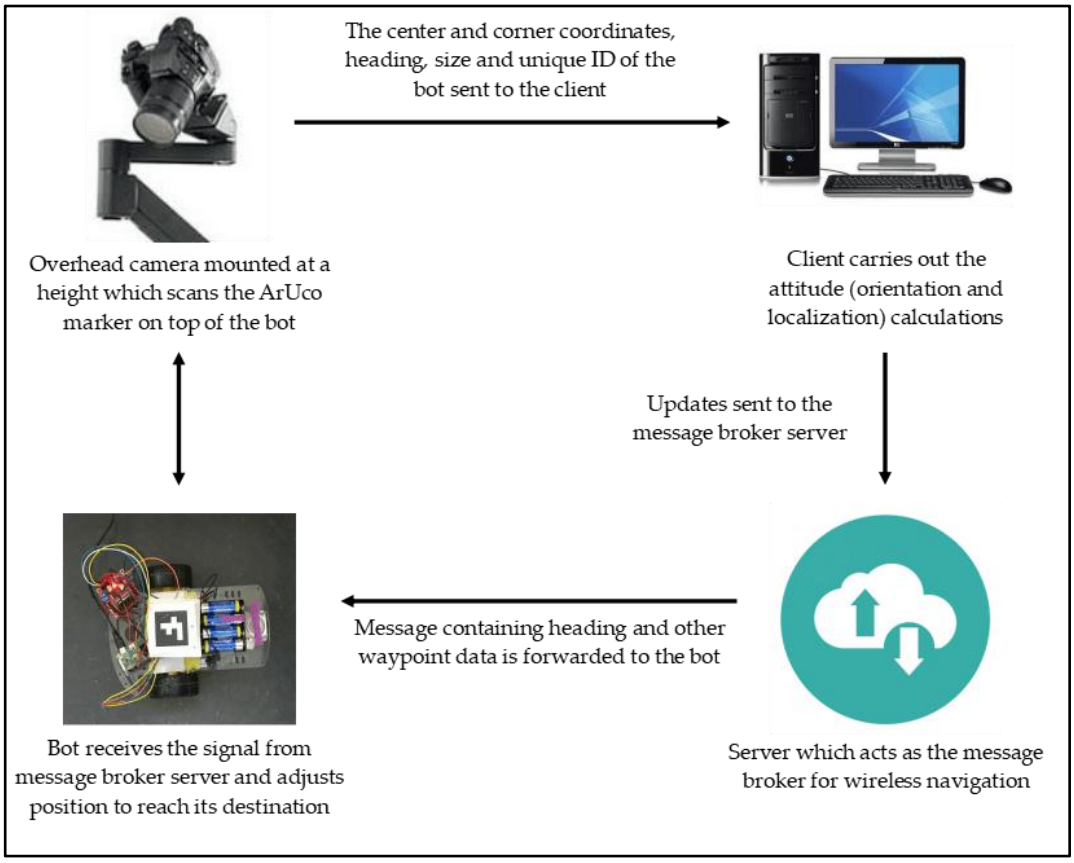

Fig. 2.1. High level abstraction of data and instruction flow among different interacting components within the project

The message broker protocol popularly used is MQTT (Message Queuing Telemetry Transport) which is a lightweight, publish-subscribe network protocol that transports messages between devices.

2.2 Project Work-flow

The following flowchart shows the detailed view of the processes happening at each step and how data and instructions are forwarded to the next entity after each subsequent step in the work-flow cycle for further processing.

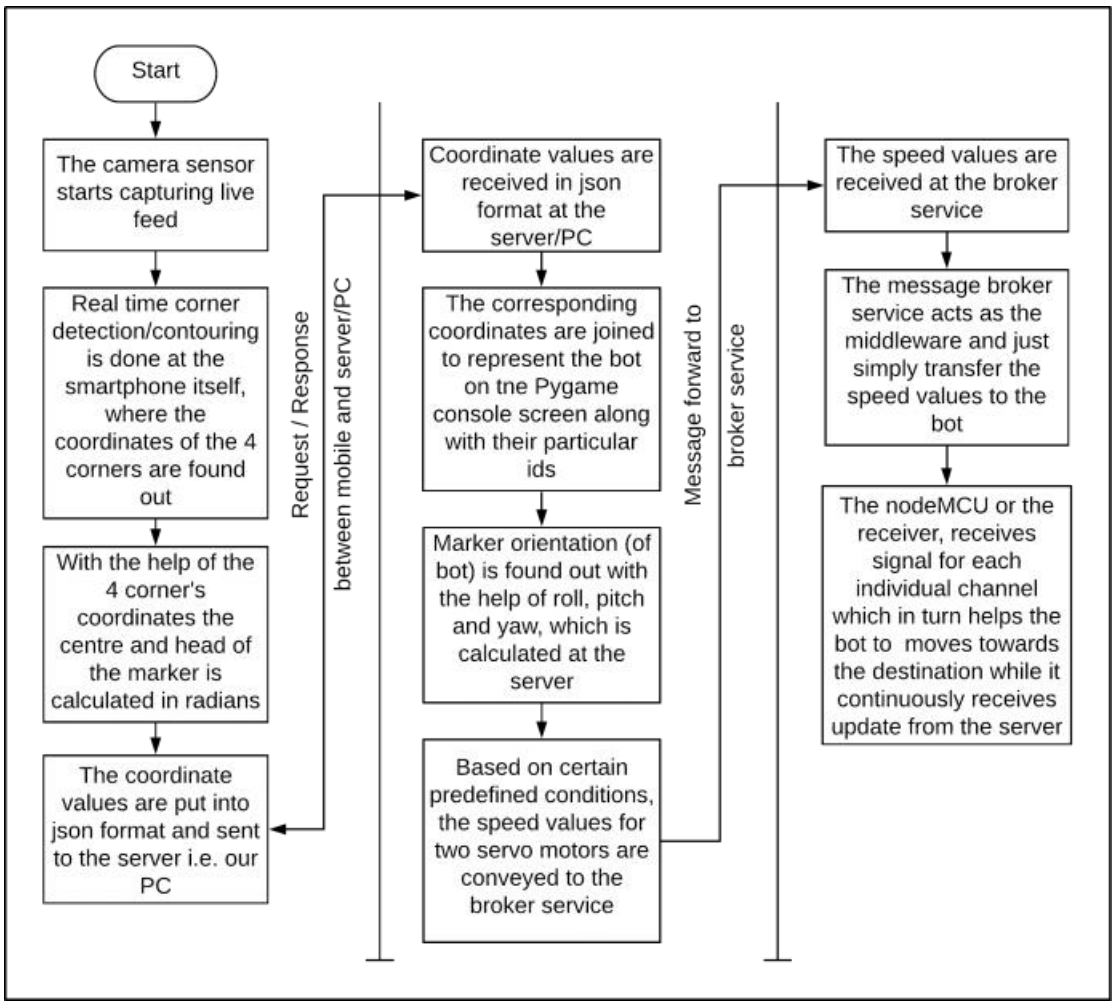

Fig. 2.2. Comprehensive view of work-flow and data exchange among different interacting components within the project

Methodologies of Implementation

This project primarily addresses the issue of indoor navigation using image processing which is implemented with the help of the OpenCV python module. In that regard, our chosen study of methods deals with the following implementation stages (with detailed algorithms and development specifications discussed later on).

The implementation of the ArUco-Nav project is achieved in the following stages -

1. Generation of ArUco markers
2. Camera calibration
3. Detection of ArUco markers -
    a) Using a standard webcam
    b) using IP webcam (Android application)
    c) using ArUco Scanner (Android application) [chosen method for this project]
4. Processing the video feed for obtaining marker orientation and simulating robot localization
5. Implement waypoint navigation between multiple targets by communicating with the swarm robots (using MQTT/TCP or other suitable protocols)

The ArUco markers used in this project have been generated using a web application that we have designed (using Bootstrap, JavaScript and jQuery). All markers belong to the original ArUco (5x5) dictionary.

OpenCV helps in successfully detecting the ArUco markers and identifying the center coordinates along with the corner values. The centre of the aruco marker is calculated at the video source and the data is relayed to the server which then calculates the distance between the x and y coordinates of ArUco marker and x and y coordinates of the target.

The main objective here is to make both the coordinates same i.e. the coordinates of an ArUco marker should overlap with that of the target. In this way we can make the bot traverse a predefined path autonomously without any human intervention.

Software and Hardware Requirements

2.3 Software Requirements

1. ArUco-Gen [web based ArUco marker generator]
2. ArUco Scanner application (for Android OS) [release used: v1.1.0]
3. Windows XP/7/10 with Python 3 installed [version used: Windows 10 64-bit with Python 3.8.2]
4. Message broker server [preferably MQTT]
5. Arduino IDE
6. Python 3 modules required:
    a) ar-markers==0.5.0
    b) numpy==1.18.3
    c) opencv-contrib-python==4.2.0.34
    d) opencv-python==4.2.0.34
    e) paho-mqtt==1.5.0
    f) simple-pid==0.2.4

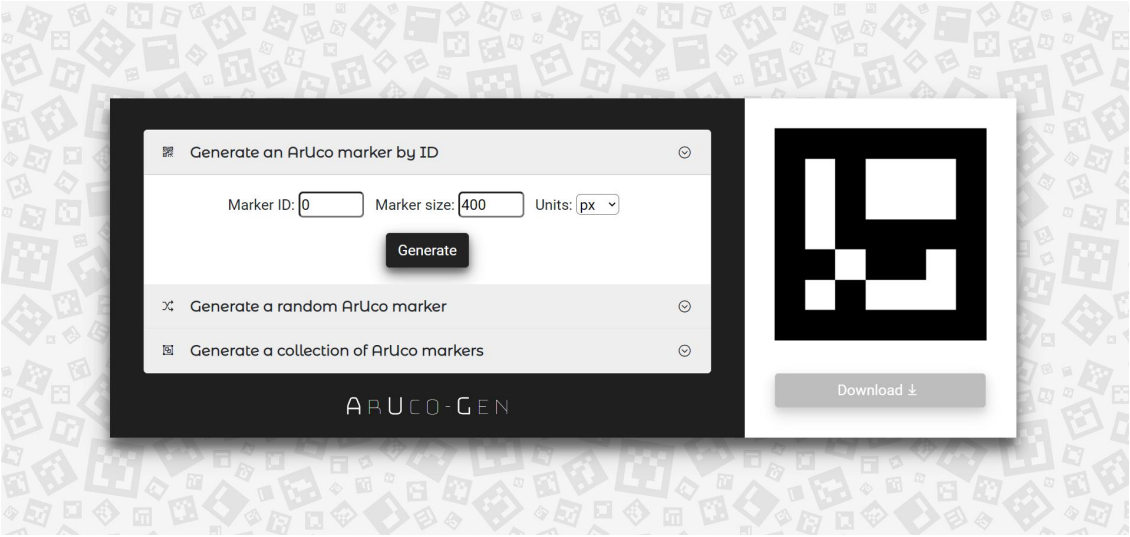

Fig. 2.3. ArUco-Gen web app for creating ArUco markers with fixed and random IDs

2.3 Hardware Requirements

1. Android smart phone with camera (at least 720p resolution)
2. Micro-mouse (or a standard 2WD robot)
3. Arduino Nano (for robot locomotion)
4. IoT development board (for wireless communication) [Bolt IoT]
5. PC/Laptop with minimum 2GB of RAM and Wi-Fi capabilities [system used: ASUS Vivobook S (X510U) with Intel Core i5-8250U x-64 based CPU and 8GB RAM]

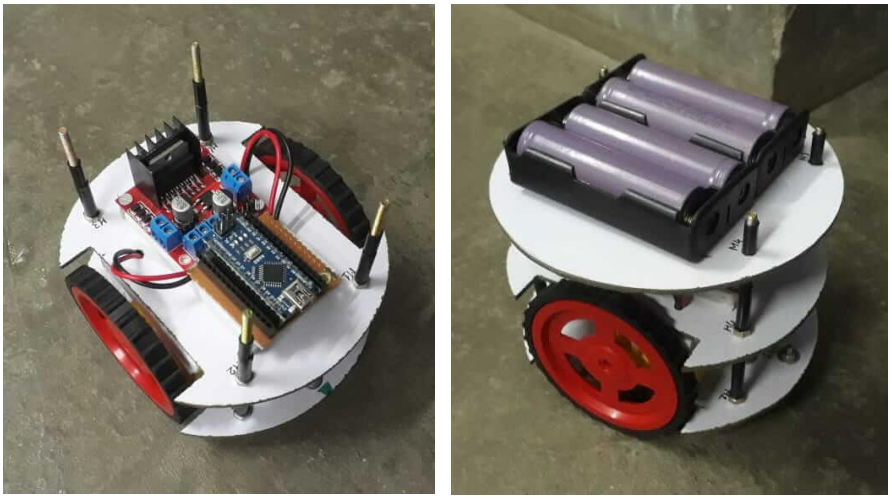

Fig. 2.4. 2WD robot chassis with Arduino Nano, L298N motor driver, BO-motors for drive and 3.7V 1500mAh Li-ion cells for powering the bot

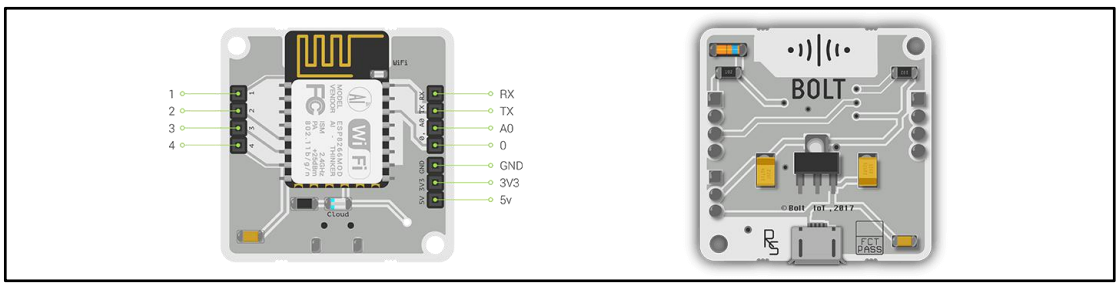

Fig. 2.5. Bolt IoT Wi-Fi module with ESP8266-12E

Our Approach

Our proposal to solve the localization and navigation problem for a robot (or a swarm of robots) is to use an overhead fiducial marker detector. Rather than using the overhead camera simply as a source of continuous video feed, we use the computing capabilities in modern day smart phones to process the video output and return marker data packaged with the response.

Our strategy to generate ArUco markers involved the use of a robust web-based marker generator that we designed using JavaScript. Instead of using simple library-based marker generators that output markers with several restrictions, we have developed an application that can generate markers having fixed or random IDs of any size.

Our approach to solve marker detection was to use an open-source project titled **Aruco Android Server** which we modified to include more specific computations pertaining to our project. The new application named **ArUco Scanner** not only scans for fiducial markers but also computes the marker ID, corner and center coordinates, heading (in radians) and the marker size.

Compared to other detection schemes like using a standard webcam or a popular Android application like IP webcam, ArUco Scanner achieves a lot more by sending a packed JSON response containing marker data which would otherwise consume unnecessary processing power on the client side.

A more detailed implementation flow has been discussed in the chapter below.