ENPM808X - Software Development for Robotics
Proposal for Vision System for MediBot
Shon Cortes
Sameer Pusegaonkar

Introduction :

MediBot: A 4 wheeled mobile robot developed by Acme Robotics has become an essential part of all hospitals across the state. This robot travels through hallways to deliver medicines reliably to their patients. Given the drastic increase in the number of patients, nurses, doctors during pandemics & flu seasons [1,2], this robot needs to make sure that it doesn't get obstructed by any humans when it travels through the hallways.

MediBot is built on an x64 architecture[3] that uses a microprocessor that runs Ubuntu 18.04 operating system[4]. We plan to implement a real-time software module for detecting & tracking humans by providing their location information with respect to the robot's coordinate frame. The robot coordinate frame is at the center of mass & the camera's coordinate frame will be at the center of the camera mounted on top of the robot. The camera system is monocular & parameters such as the Field Of View, distortion coefficients, frame rate & its position are known. All other calculations will be done in SI units.

This video input from the camera will be processed by a machine-learning model to detect humans. Corresponding location information can be then found out with respect to its camera frame. Since the robot is given input in terms of x & y coordinates, we will transform these coordinates into the robot's reference frame. The z coordinates are ignored since we only have a monocular camera. Once that is done this information can then be used by Acme Robotics to understand where humans are with respect to MediBot's position & eventually avoid them.

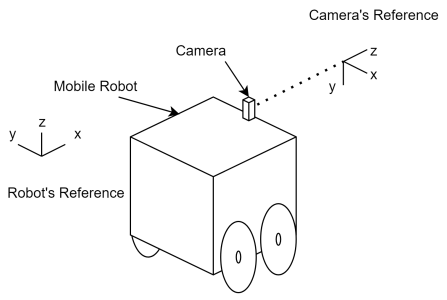


Figure 1: MediBot

Project Organization:

The Agile development process will be used to ensure the quality of work and task tracking. Our team consists of two people, Sameer Pusegaonkar & Shon Cortes, expecting to work 10 hours during each one-week sprint. This means our iteration capacity is 20 hours. Product backlogs[Fig. 2], sprint progress, bugs fixes, time logs, and test plans will be tracked using Azure DevOps.

An initial UML activity and class diagram, [Fig. 3, 4], has been made outlining the flow of the program which has fed our initial product backlog. During each sprint, bug catches and new features will be tracked and added to the backlog to be handled during the next sprint. Daily scrum meetings will be held to discuss progress and address sticking points. We will use pair programming to assist in solving any problems an individual team member encounters as needed. At the end of each sprint, the code and product backlog will be evaluated during an iteration review in preparation for the next sprint. Unit tests will be designed using the google test library to ensure full code coverage and proper operation. Git will be the primary tool for version control and tracking.

Managerial Process:

Testing will be done on the entire codebase. This includes unit testing, integration testing & regression testing. V-Model will be followed for the development & testing of our code. To test the accuracy of the model we will be collecting pre-labeled data. The model will only be accepted if it has an accuracy above 80%. A backup plan to implement a color tracker or template matching will be set up if the above technique falls below the 80% desired threshold accuracy.

A potential flaw that could come up is that because of the constraint of a monocular camera, a large tolerance in position will be noticeable due to the use of the average height of humans. Variations in lighting conditions can also cause errors in position estimation as well as increase the difficulty in obstacle detection. Ideally, depth estimation would be done with a stereo vision system using two cameras. In this application, we are restricted to a singular monocular camera system which may increase our margin of error as well. Our timeline is ambitious given we are a small team and will be working under short sprints.

Technical Process:

To get the location information of humans, we will be utilizing a pre-trained model called mobilenet[5] which will allow us to get bounding boxes for detected objects. MobileNet is based on a streamlined architecture that uses depth-wise separable convolutions to build lightweight deep neural networks[5]. Because of the constraints of the robot hardware, mobile net was picked as it achieves comparable results to other networks like RCNN, Faster CNN with only a fraction of computationally complexity and model size[5]. This model outputs the bounding boxes for N>=1 humans. A correlation filter will be used to track each and every one of these instances of humans using DLib[6].

Once that information has been obtained in the camera's reference frame, this information could be transformed into the robot's reference frame. This process is done by computing a similarity triangle. Since we are aware of the focal length, field of view of the camera, and the width and height of the humans in pixels, this information can be converted into the x coordinate for the robot which is the z coordinate for the camera frame. Using the depth information, we can determine the human's y position (horizontal position in the camera frame) by mapping horizontal pixel locations to physical horizontal positions. While this is an estimation for the actual x and y coordinates, the method is robust for the purposes of Acme's Medibot.

Work Packages:

C++ 11 and above [7] will be the primary language for the entire codebase. We will be utilizing Cmake version 3.2.1 and above [8] for our build system. For processing our video input, OpenCV version 4.5.6 and above (Apache 2 license) [9] will be used & DLib version 19.22 and above (Boost Software License - Open Source)[6] for implementing the tracking functionality. A MobileNets [5] pre-trained model will be used for human detection.

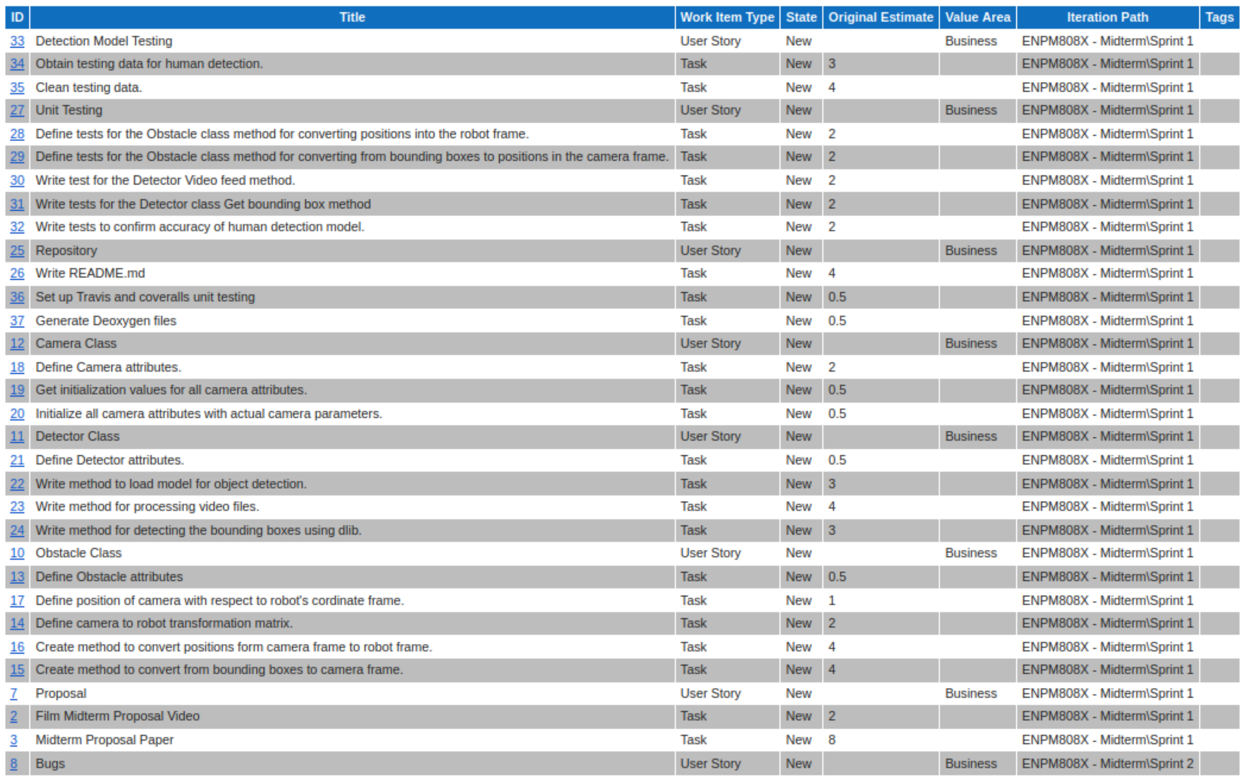
At the end of this contract, Acme Robotics will receive a well-documented & fully tested software module that can detect and track humans while returning their position in the robot frame.

[1] https://ourworldindata.org/covid-hospitalizations

[2] https://www.cdc.gov/coronavirus/2019-ncov/covid-data/covidview/index.html

[3]https://en.wikipedia.org/wiki/X86-64
[4] https://releases.ubuntu.com/18.04/
[5] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications - https://arxiv.org/abs/1704.04861

[6] http://dlib.net/imaging.html
[7] https://isocpp.org/
[8] https://cmake.org/
[9] https://github.com/opencv/opencv



ID	Title	Work Item Type	State	Original Estimate	Value Area	Iteration Path	Tags
33	Detection Model Testing	User Story	New		Business	ENPM808X - Midterm\Sprint 1	
34	Obtain testing data for human detection.	Task	New	3		ENPM808X - Midterm\Sprint 1	
35	Clean testing data.	Task	New	4		ENPM808X - Midterm\Sprint 1	
27	Unit Testing	User Story	New		Business	ENPM808X - Midterm\Sprint 1	
28	Define tests for the Obstacle class method for converting positions into the robot frame.	Task	New	2		ENPM808X - Midterm\Sprint 1	
29	Define tests for the Obstacle class method for converting from bounding boxes to positions in the camera frame.	Task	New	2		ENPM808X - Midterm\Sprint 1	
30	Write test for the Detector Video feed method.	Task	New	2		ENPM808X - Midterm\Sprint 1	
31	Write tests for the Detector class Get bounding box method	Task	New	2		ENPM808X - Midterm\Sprint 1	
32	Write tests to confirm accuracy of human detection model.	Task	New	2		ENPM808X - Midterm\Sprint 1	
25	Repository	User Story	New		Business	ENPM808X - Midterm\Sprint 1	
26	Write README.md	Task	New	4		ENPM808X - Midterm\Sprint 1	
36	Set up Travis and coveralls unit testing	Task	New	0.5		ENPM808X - Midterm\Sprint 1	
37	Generate Deoxygen files	Task	New	0.5		ENPM808X - Midterm\Sprint 1	
12	Camera Class	User Story	New		Business	ENPM808X - Midterm\Sprint 1	
18	Define Camera attributes.	Task	New	2		ENPM808X - Midterm\Sprint 1	
19	Get initialization values for all camera attributes.	Task	New	0.5		ENPM808X - Midterm\Sprint 1	
20	Initialize all camera attributes with actual camera parameters.	Task	New	0.5		ENPM808X - Midterm\Sprint 1	
11	Detector Class	User Story	New		Business	ENPM808X - Midterm\Sprint 1	
21	Define Detector attributes.	Task	New	0.5		ENPM808X - Midterm\Sprint 1	
22	Write method to load model for object detection.	Task	New	3		ENPM808X - Midterm\Sprint 1	
23	Write method for processing video files.	Task	New	4		ENPM808X - Midterm\Sprint 1	
24	Write method for detecting the bounding boxes using dlib.	Task	New	3		ENPM808X - Midterm\Sprint 1	
10	Obstacle Class	User Story	New		Business	ENPM808X - Midterm\Sprint 1	
13	Define Obstacle attributes	Task	New	0.5		ENPM808X - Midterm\Sprint 1	
17	Define position of camera with respect to robot's cordinate frame.	Task	New	1		ENPM808X - Midterm\Sprint 1	
14	Define camera to robot transformation matrix.	Task	New	2		ENPM808X - Midterm\Sprint 1	
16	Create method to convert positions form camera frame to robot frame.	Task	New	4		ENPM808X - Midterm\Sprint 1	
15	Create method to convert from bounding boxes to camera frame.	Task	New	4		ENPM808X - Midterm\Sprint 1	
7	Proposal	User Story	New		Business	ENPM808X - Midterm\Sprint 1	
2	Film Midterm Proposal Video	Task	New	2		ENPM808X - Midterm\Sprint 1	
3	Midterm Proposal Paper	Task	New	8		ENPM808X - Midterm\Sprint 1	
8	Bugs	User Story	New		Business	ENPM808X - Midterm\Sprint 2	

Figure 2: Product Backlog

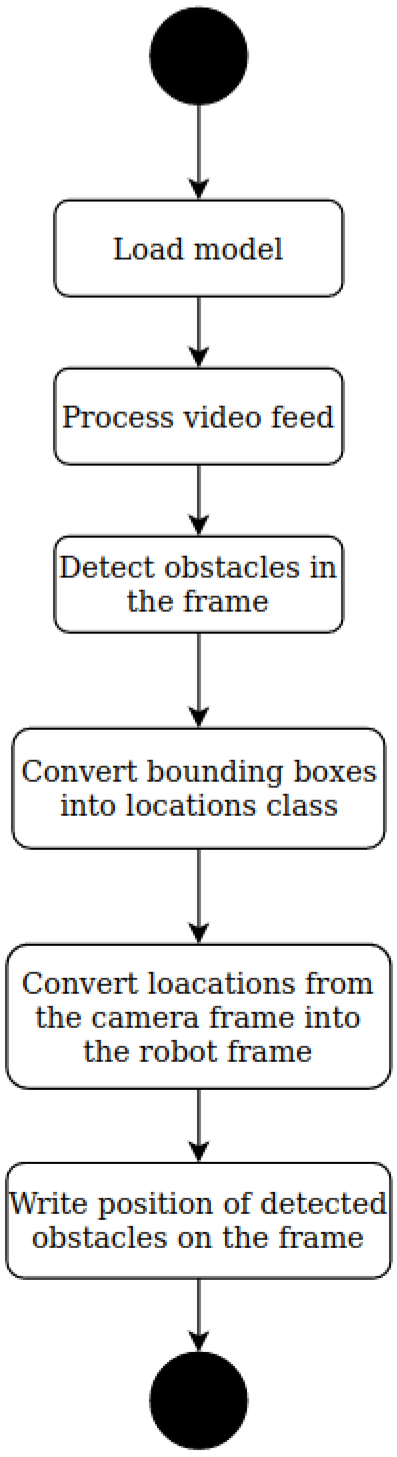

Load model
Process video feed
Detect obstacles in the frame
Convert bounding boxes into locations class
Convert loacations from the camera frame into the robot frame
Write position of detected obstacles on the frame

Figure 3: Activity Diagram

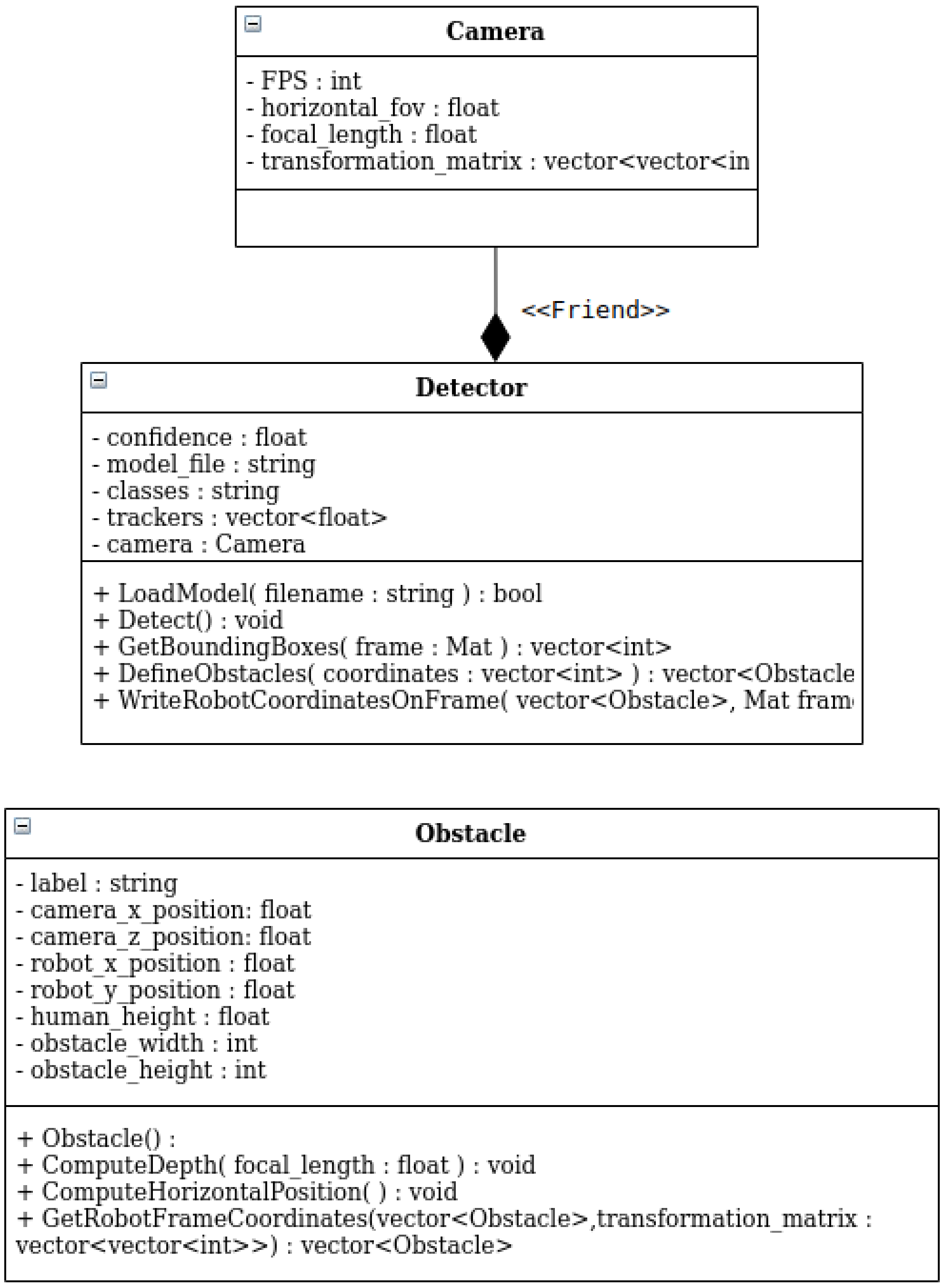

Camera
- FPS : int
- horizontal_fov : float
- focal_length : float
- transformation_matrix : vector<vector<in
<<Friend>>
Detector
- confidence : float
- model_file : string
- classes : string
- trackers : vector<float>
- camera : Camera
+ LoadModel(filename : string) : bool
+ Detect() : void
+ GetBoundingBoxes(frame : Mat) : vector<int>
+ DefineObstacles(coordinates : vector<int>) : vector<Obstacle
+ WriteRobotCoordinatesOnFrame(vector<Obstacle>, Mat fram
Obstacle
- label : string
- camera_x_position: float
- camera_z_position: float
- robot_x_position : float
- robot_y_position : float
- human_height : float
- obstacle_width : int
- obstacle_height : int
+ Obstacle() :
+ ComputeDepth(focal_length : float) : void
+ ComputeHorizontalPosition( ) : void
+ GetRobotFrameCoordinates(vector<Obstacle>,transformation_matrix : vector<vector<int>>) : vector<Obstacle>

Figure 4: Class Diagrams

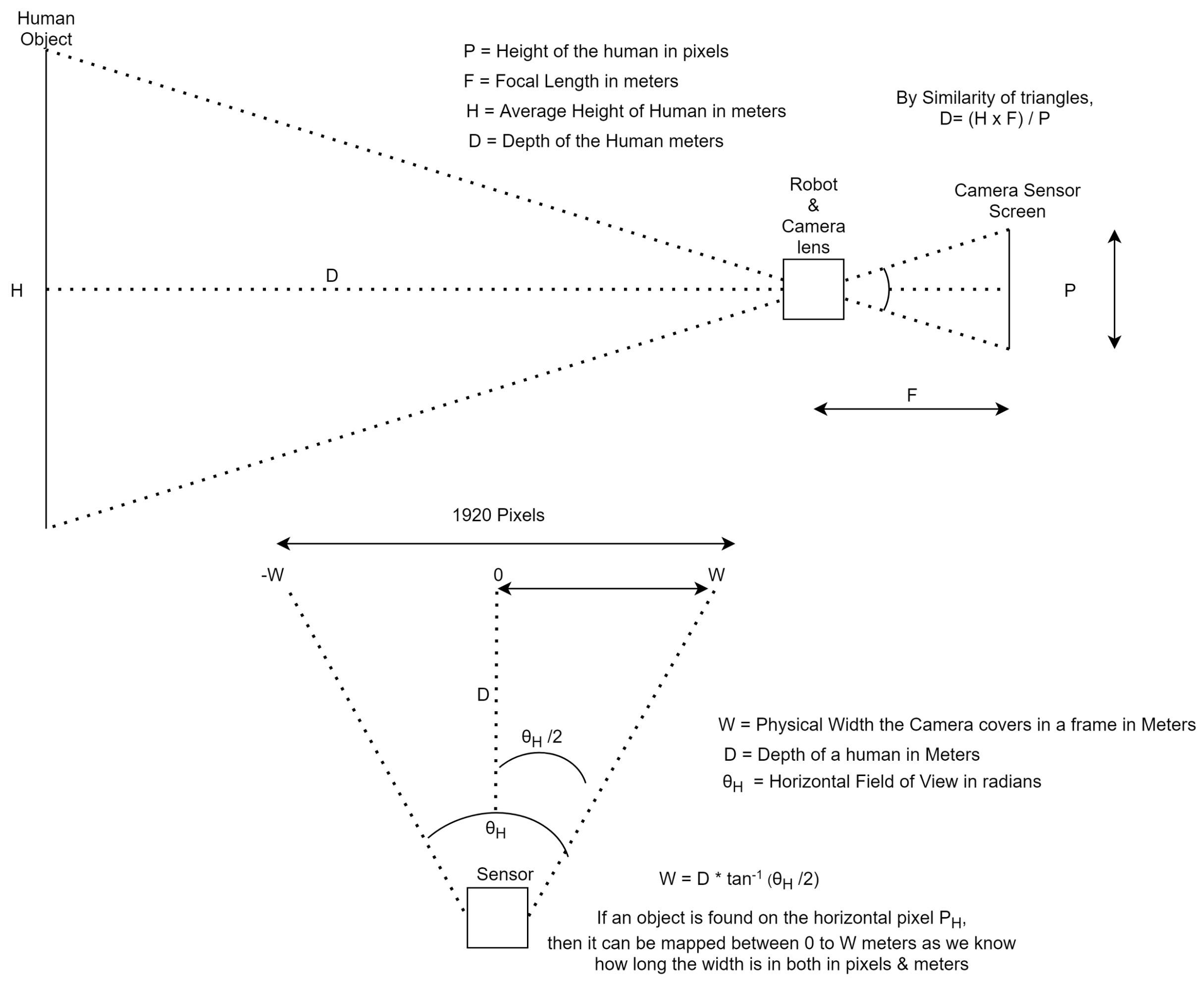

Human Object
P = Height of the human in pixels
F = Focal Length in meters
H = Average Height of Human in meters
D = Depth of the Human meters
By Similarity of triangles, D= (H x F) / P
Robot & Camera lens
Camera Sensor Screen
H
D
P
F
1920 Pixels
-W
0
W
D
θ$_H$ /2
θ$_H$
Sensor
W = Physical Width the Camera covers in a frame in Meters
D = Depth of a human in Meters
θ$_H$ = Horizontal Field of View in radians
W = D * tan^{-1} (θ$_H$ /2)
If an object is found on the horizontal pixel P$_H$, then it can be mapped between 0 to W meters as we know how long the width is in both in pixels & meters

Figure 5: Position calculation diagrams