Komodo robot project summary

Team members:

Ehud Barnea, 300021011, [barneaeh@cs.bgu.ac.il](mailto:barneaeh@cs.bgu.ac.il)

Boaz Arad, 200134252, [Boazar@bgu.ac.il](mailto:Boazar@bgu.ac.il)

Rotem Mairon, 039781729, [rotemra@bgu.ac.il](mailto:rotemra@bgu.ac.il)

# Project goals

The main purpose of our project is to build a basic framework for collision free movement of the robot in the presence of obstacles. Additionally, we sought to provide a simple form of robot interaction with humans in the environment. In order to achieve the above, the following issues were tackled:

1. Sensor integration:

While robot control can be simplified to a continuous “input->processing->action” loop, processing multiple asynchronous sensory inputs adds significant complexity to this process. Here we examine how different sensory input from the environment should be combined in order to allow the robot to take desired action in a timely manner.

1. Collision prevention:

Collision free movement in the presence of obstacles requires safety mechanisms which to prevent the robot from colliding with dynamic obstacles. In our project, we use close-range sensory input to implement such a safety mechanism.

1. Obstacle avoidance:

In addition to avoiding dynamic obstacles, continuous movement of the robot requires planning in order to avoid stationary obstacles. To this end, we rely on mid-range sensory input.

1. Interaction with human environment:

As part of this project we implement people detection as a high level capability to interact with humans in the environment in parallel with the movement of the robot.

Below we describe how various sensory inputs were combined in order to implement the above capabilities.

# Sensor integration and control module

An essential issue in designing a robot that acts in the presence of obstacles is the integration of different sensory input. Information from different sensors is transmitted asynchronously. In order to enable the robot to take proper actions according to this information, at any given time, we must combine sensory information into a single state. In our project information from the following sensors is combined:

1. Frontal camera
2. Left range sensor
3. Right range sensor
4. Frontal laser scanner

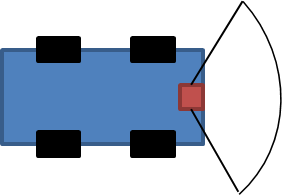
In order to process the sensory input, each sensor is assigned with a callback function. Each function is executed whenever additional sensory input is acquired. Since all different sensors operate in parallel, each callback function is in a different thread. In order to avoid data corruption all threads must be properly synchronized.

The information from different threads is combined in the function *motionControl* (see file rob.cpp, line 174). Based on the combined information, this function controls the actions taken by the robot. E.g., the callback function associated with the frontal laser scanner stores information such as whether or not the robot is blocked by some object, or the suggested angle of a turn the robot should take (to avoid an obstacle before reaching it). Based on this (and other) information, the motion control procedure determines the proper action and performs it. In our case, the action is either "stop", or "move in a certain direction" (see file rob.cpp, lines 200-206).

# Collision prevention

Obstacles may either be part of the environment in which the robot acts, or unexpectedly occur (e.g., people walking, doors opening, furniture moving etc.). Collision prevention procedures allow the robot to avoid both expected and unexpected obstacles.

Our strategy is to allow the robot to advance only after ensuring that a collision will not occur due to this motion. This is achieved using frontal laser sensor of the robot. At all times, this sensor provides indication of any potential obstacle which is too close to the robot (see Figure 1). Using the laser sensor we ensure a safety area in radius of 40 centimeters ahead of the robot and an opening angle of 120 degrees. As long as the safety area is clear, the robot may advance. Otherwise, it halts.



Figure

**Code details**:

The collision prevention is implemented in the main source file (*rob.cpp*). The extent of the safety area is defined in the same file by the variables *safeDistance* (line 146) and *safetyAngle* (line 147). The callback function *LaserScan\_callback* (line 122) is executed continuously and responds to any indication by the frontal laser sensor of an obstacle in the defined safety area. The operation of the relevant callback is based on the following:

1. Determine the safety zone:

The input from the laser sensor is stored in "range pixels". Each range pixel observes a certain angular range. The value stored in *safetyAngle* is converted into the number of range pixels that are relevant to verify a clear safety zone (line 148).

1. Verify the safety zone:

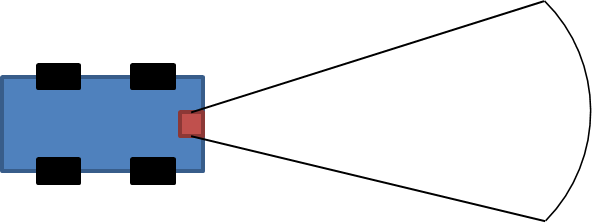
The function *canAdvance* (see line 53) uses the information stored in the selected range pixels as input. Based on this input the function verifies that any object indicated by relevant range pixels is indeed further away from the threshold value stored in *safeDistance*.

1. Take proper action:

The output from the function *canAdvance* is stored in the variable *can\_move* in a synchronized manner. Therefore, the state of this variable indicates whether the safety zone is clear or not.

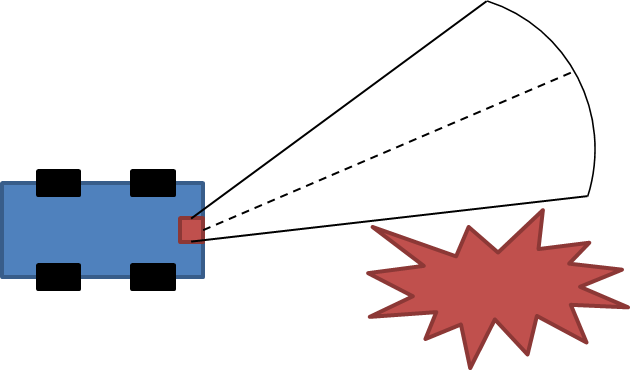
# Obstacle avoidance

The collision prevention mechanism described above is mainly intended to enable the robot to react (stop) in case of an unexpected obstacle. The obstacle avoidance module is responsible for determining the course of the robot for a longer term. This is done by acquiring range information from further ahead in order to avoid existing obstacles, without having to stop.



Figure

Using the laser sensor we verify that a "corridor" in radius 1.5 meters ahead of the robot, at an opening angle of 40 degrees (see Figure 2), is clear from obstacles. In case it is, the robot may keep on moving forward. Otherwise, the area in front of the robot (within the same corridor) is scanned find an alternative corridor, such that clear driving paths may be found either to the right or to the left direction, as can be seen in Figure 3. In case several alternatives corridors are detected, the one closest to the center of the robot is picked.



Figure

**Code details**:

The obstacle avoidance module is implemented in the main source file (rob.cpp). It receives input from the callback function *LaserScan\_callback* (line 122). The size of the corridor is stored in the variables *numCorridorPixels* (lines 161) and *clearDistance* (lines 162). The operation of the relevant callback is based on the following:

1. The function *findClearCorridor* (line 74) examines all possible corridors of fixed size (same opening angle and distance) in order to find clear corridors in which the robot could advance.
2. From the clear corridors, *findClearCorridor* selects the one which is most aligned with the front axis of the robot. The angle between the central axis of the corridor (dashed line in Figure 3) and the front axis of the robot is stored in the variable *direction* in a synchronized manner.
3. The value stored in *direction* is used to determine the required direction and angle in which the robot should turn (right / left).

# People detection

A robot that interacts in a human-like manner would exhibit an impression of intelligent behavior and thus facilitate interaction with people. To display the potential for such ability, we have equipped it with face detection capabilities. This allows the robot to detect people and address them while it advances, avoiding obstacles.

**Code details**:

In parallel with the constant operation of the movement modules, the robot processes visual information acquired by the frontal camera searching for faces. The face detection module is implemented in the file *FaceDetector.cpp* (and its header file, *FaceDetector.h*) which allows easy reuse of the code in other projects.

The rob.cpp file constantly “listens” to visual input (color images) from the front camera of the robot and calls the function *detect* (located in FaceDetector.cpp) for each image. This function receives the input image and outputs the locations of any detected faces. In order to ensure robust detection, we require that a face is detected in a number of consecutive frames. Once this condition holds, the robot is “confident” in the presence of a person and greets them by playing a special sound sample. The implementation is based on the following:

1. Whenever a new image is acquired the function *frontCam\_Callback* (line 211 in rob.cpp) is called and wraps the input in a ROS message.
2. The image is then extracted from within the ROS message and converted into the OpenCV image format using cv\_bridge (line 216).
3. Next, a face detector object detects any faces in the extracted image (line 220).
4. Whenever a face is detected, the value of the counter variable *face\_detections* is updated (increased) in a synchronized manner. When a face has been detected a fixed number of times, a greeting sample sound is played and the control module is updated to indicate that a person has been greeted.

# Summary

We have implemented a simple architecture that allows a robot to move in an unknown environment. Our implementation exploits the sensory input for avoiding both existing and unexpected obstacles. In addition, the implementation allows a simple interaction with human environment, by detecting people and greeting them. During our experiments the robot was indeed able to drive autonomously and react to dynamic conditions of the environment. These capabilities are demonstrated in the supplied video.