Collision avoidance with a LEGO-car

Hannes Lindstrand  Pierre Höglin
Sofia Södergren  Daniel Salminen

March 15, 2013

Abstract

The objective of this project was to develop methods for collision avoidance for a mobile LEGO robot with on-board sensors. The robot was three wheeled and has two rotating sensors at the front, giving information about the surrounding environments. To detect the distance to an obstacle two different kinds of sensors were used, ultrasonic distance sensors and IR distance sensors. Five different models were developed, two using ultrasonic sensors, two using IR sensors and one using both ultrasonic sensors and IR sensors. The potential field method were used for the solutions with IR sensors and distance limits were used for the solutions with ultrasonic sensors.
## Contents

1 Introduction .................................................. 3

2 Theory .................................................................. 4
  2.1 LEGO MINDSTORMS ........................................ 4
    2.1.1 NXT ..................................................... 4
    2.1.2 Ultrasonic Sensor ....................................... 4
    2.1.3 IR Distance Sensor ..................................... 4
    2.1.4 nxtoSEK .................................................. 4
  2.2 Collision avoidance methods .............................. 5
    2.2.1 Potential Field method .................................. 5
    2.2.2 Vector field Histogram .................................. 6
    2.2.3 Dynamic Window Approach ............................ 6

3 Implementation .................................................. 7
  3.1 Robot Construction .......................................... 7
    3.1.1 Ultrasonic Sensor Robot ............................... 8
    3.1.2 IR Sensor Robot ......................................... 9
    3.1.3 IR Sensor and Ultrasonic Sensor Robot .......... 9
  3.2 Implementation with Ultrasonic Sensors .............. 10
    3.2.1 Model I .................................................. 10
    3.2.2 Model II ................................................. 10
  3.3 Implementation with Ultrasonic Sensors and IR sensor 12
  3.4 Implementation with IR Sensors ........................ 12
    3.4.1 Model I .................................................. 13
    3.4.2 Model II .................................................. 15
  3.5 Programming issues and recommendations .............. 15

4 Result .................................................................. 16
  4.1 Result with Ultrasonic Sensors .......................... 16
    4.1.1 Model I .................................................. 16
    4.1.2 Model II .................................................. 16
  4.2 Result with Ultrasonic Sensors and IR Sensors .... 17
  4.3 Result with IR Sensors ..................................... 17
    4.3.1 Model I .................................................. 17
    4.3.2 Model II .................................................. 17

5 Conclusions ..................................................... 18

6 References ....................................................... 20
1 Introduction

One of the most important things for anything that moves in an unknown environment is to have a good collision avoidance system (CAS). There are many variants of CA systems used in many forms of vehicles. The most basic CAS is to just look out of a window and turn or break if needed, this obviously only works when we have windows and a human driver. Other CA systems can be used for example in cars, which use sensors that look in front of the vehicle and breaks if a crash is imminent, so far this only works in low velocities. Ships and aircraft have a system that inter alia acts as a CAS. Both ships and aircraft have a transponder that sends position, velocity, name of the vehicle and other information. The signal is sent to a flight control or vessel control service, they can detect if the vehicles is on a colliding course. Robots have a CAS that need to be a little more advanced since it does not have a human driver. Therefore they need to look around a lot more and have a good solution on how to act if it detects an obstacles.

This project will focus on systems implemented in robots and build models from the LEGO mindstorms set with different approaches. There are existing methods for this and we will implement two of them and vary the implementations depending on how the robot is constructed. To our help we have infra-red sensors for a narrow but precise measure and ultrasonic distance sensors for a more wide and less precise measure.

There are two different kinds of systems used in robots; global and local methods. Global methods focus on making a map of the environment, when that is done it can easily find an optimal route to its target. The downside with these systems is that they are generally slow when building a map and they have problems when the environment changes.

Local methods use the robots surrounding environment at the moment and determines path for a short time ahead, then it does the same step again and so on. These methods are faster and handles a dynamic environment better. The problem with these methods is that they can’t find an optimal route and can get stuck. Local methods will be used in this project.

In order to organize the project we split the work between each other. Sofia wrote most of the report. Daniel helped with the report, made all figures and was the project leader. Pierre and Hannes wrote most of the code since they had most experience in that. Hannes also made the films. We all contributed in the presentation and our discussions.
2 Theory

In this section the theory considered in this project is presented.

2.1 LEGO MINDSTORMS

LEGO MINDSTORMS is a series of sets consisting of software and hardware, that can be used to create programmable and customizable robots. A set consists of computer (NXT), sensors, motors and building blocks (LEGO).

2.1.1 NXT

The MINDSTORM robots brain is the NXT. The NXT is an intelligent computer-controlled LEGO block, through which the robot can perform different operations. On the NXT there are three output ports for motors, four inputs for attaching sensors and one USB port. The NXT has a display with a various of different features. [1]

2.1.2 Ultrasonic Sensor

The ultrasonic sensor measures distances and thereby makes it possible for the robot to perceive objects. The measurements made are in centimetres, with a precision of +/- 3 centimetres. The ultrasonic sensor can detect objects on a distance from 0 to 255 centimetres. The ultrasonic sensor uses the time it takes for a sound wave to reach an object and return, to find the distance from the robot to the object. [1]

2.1.3 IR Distance Sensor

The Medium IR sensor can detect objects on a distance of 10 to 80 centimetres and the Short IR sensor can detect objects on a distance of 6 to 30 centimetres, measuring the distances in millimeters. A typical response time for an IR sensor is 39 milliseconds. [2]

2.1.4 nxtOSEK

The nxtOSEK is used for the LEGO MINDSTORMS and is a real-time operating system (RTOS). To program the robots when using nxtOSEK, ANSI-C or C++ can be used as programming language. [3]
2.2 Collision avoidance methods

For this project, local collision avoidance methods that use the current measurement to decide the further direction and speed are considered. In this section three methods are presented, potential field method, vector field histogram and dynamic window approach.

2.2.1 Potential Field method

The potential field method is based on the idea of imaginary forces acting on a robot (suggested by Andrews and Hogan, 1983; Khatib, 1985). The environment is described as a vector field, where obstacles act on the robot as a repulsive forces and a target acts as an attractive force. The repulsive forces are defined as

\[
\vec{F}_{i,j} = \frac{F_{cr} W^n C_{i,j}}{d^n(i,j)} \left( \frac{x_i - x_0}{d(i,j)} \hat{x} + \frac{y_j - y_0}{d(i,j)} \hat{y} \right)
\]

where \(F_{cr}\) is the repulsive force constant, \(d(i,j)\) is the distance between the active cell \((i,j)\) and the robot, \(C_{i,j}\) is the certainty value of the active cell \((i,j)\), \(W\) is the width of the robot, \((x_0, y_0)\) is the robots present coordinates, \((x_i, y_j)\) is the coordinates of the active cell \((i,j)\) and \(n\) is a positive constant. The resultant repulsive force is given by

\[
\vec{F}_r = \sum \vec{F}_{i,j}
\]

and the attractive force are defined as

\[
\vec{F}_a = F_{ct} \left( \frac{x_t - x_0}{d_t} \hat{x} + \frac{y_t - y_0}{d_t} \hat{y} \right)
\]

where \(F_{ct}\) is the attraction force constant from the target, \((x_t, y_t)\) is the targets position and \(d_t\) is the distance from the robot to the target.

The sum of all these forces

\[
\vec{F}_{sum} = \vec{F}_a + \vec{F}_r
\]

determines the further direction and speed. The speed is determined from the magnitude of \(\vec{F}_{sum}\) and the direction from the direction of \(\vec{F}_{sum}\). [4], [5], [6], [7].
2.2.2 Vector field Histogram

The vector field Histogram method was proposed by Johann Borenstein and Yoram Koren in 1991. The method is a real-time motional planning algorithm that takes the dynamics and shape of the robot into account. The model of the environment is updated continuously with data from a range of sensor on-board the robot. The method consists of a three-stage ongoing process, where the two last stages are data reduction stages. The stages are:

I) Cartesian Histogram: A two-dimensional Cartesian histogram grid is calculated from the data of the on-board sensors.

II) Polar Histogram: The histogram grids are reduced into a one-dimensional polar histogram of the robots momentary surrounding. A sector in the polar histogram is representing the polar obstacle density in that direction.

III) Select Direction: The most suitable sector is selected. Thus, the most suitable sector among all sectors with low polar obstacle density, a polar obstacle density under the selected threshold. The robot is then guided in the direction of the most suitable sector. [8]

2.2.3 Dynamic Window Approach

The Dynamic window approach is derived from the dynamics of the robot. Therefore it is well suited to deal with constraints in velocities and accelerations of the robot. The method has two stages in its control circuit, first it generates a valid search space, and secondly it determines an optimal solution in the search space.

The generation of the search space can be reduced to three steps:

I) Circular trajectories: Since we are working directly in velocity space, the speed of the robot is written in translational and rotational velocity. Therefore the method only considers circular trajectories, which results in a two-dimensional velocity search space.

II) Admissible velocities: A velocity pair \((v, w)\) is considered admissible, if the resulting curvature doesn’t result in a collision with an object.

III) Dynamic window: Due to limitations in acceleration of the robot...
the dynamic window makes sure that the admissible velocities only include those that can be reached within a short time step. The optimization is done through maximizing the objective function. Which is described as

\[ G(v, \omega) = \sigma(\alpha \cdot \text{heading}(v, \omega) + \beta \cdot \text{dist}(v, \omega) + \gamma \cdot \text{vel}(v, \omega)) \]

\(\alpha, \beta\) and \(\gamma\) is user determined parameters, which determines the trade off for the different functions:

**Target heading:** \(\text{heading}\) measures the progress towards the target. It is at its maximum when the robot is moving directly towards the goal location.

**Clearance:** \(\text{dist}\) measures the distance to the closest object on the trajectory. When the distance is smaller, the desire to move around it is higher.

**Velocity:** \(\text{vel}\) measures the forward velocity.

The translational and rotational velocity that maximizes the objective function is then chosen as the new velocity for the robot.

The admissible velocities in a window are defined as: [9]

\[ V_a = \left\{ (v, \omega) | v \leq \sqrt{2 \cdot \text{dist}(v, \omega) \cdot \dot{v}_b} \land \omega \leq \sqrt{2 \cdot \text{dist}(v, \omega) \cdot \dot{\omega}_b} \right\} \]

### 3 Implementation

In this section the different implementation models for this project are described.

#### 3.1 Robot Construction

The robot constructed for this project consisted of one NXT, three motors, three wheels, two sensors (ultrasonic/IR) and LEGO pieces. There are two major wheels, each one driven by a motor. The small wheel is for balancing the robot. The sensors are placed in the front of the robot and are driven by the third motor.
The construction with ultrasonic sensors (to the left in the figure) and IR sensors (to the right in the figure) can be seen in Figure 1.

![Robot construction with ultrasonic sensors and IR sensors](image)

Figure 1: Robot construction with ultrasonic sensors and IR sensors

### 3.1.1 Ultrasonic Sensor Robot

The sensors are placed in the front of the robot. They are measuring distance at two different positions, $\alpha$ and $\beta$, as can be seen in Figure 2. Both sensors are in the position of the angle $\alpha$ respective $\beta$ at the same time.

![Ultrasonic sensors position](image)

Figure 2: Ultrasonic sensors position
3.1.2 **IR Sensor Robot**

The IR sensors are placed in the front of the robot. They are measuring distance at the different positions, $\alpha$, $\beta$, $\theta$ and $\gamma$ as can be seen in Figure 3. Both sensors are in the position of the angle $\alpha$, $\beta$, $\theta$ respective $\gamma$ at the same time.

![Figure 3: IR sensors position](image)

3.1.3 **IR Sensor and Ultrasonic Sensor Robot**

The IR sensor are placed in the middle in the front of the robot. The ultrasonic sensors are placed in the front of the robot. They are measuring distance at two different positions, $\alpha$ and $\beta$, as can be seen in Figure 4. Both sensors are in the position of the angle $\alpha$ respective $\beta$ at the same time.

![Figure 4: IR sensors position](image)

3.2 **Implementation with Ultrasonic Sensors**

For implementation using ultrasonic sensors two different models where used as described in the two following sections.
3.2.1 Model I

In this model, the robot keeps constant speed until it observes an object closer than or on the shortest allowed distance. To avoid the observed object, one wheel maintains constant speed and one stands still, so that the robot starts turning. This is done until the object is out of the robot’s field of view. The flowchart over the program for this model can be seen in Figure 5.

![Flowchart for Model I](image)

Figure 5: Flowchart over the program for model I using ultrasonic sensors

3.2.2 Model II

This model uses three different distance limits to determine if the robot should turn, move faster/slower or turn on the spot. The shortest distance limit (limit 1) is set to a small value and the furthest distance limit (limit 2) is set to a large value. The speed and move control at the three different distance limits are:

- **Closer than Limit 1**: If the robot is inside this distance limit it has to turn on the spot.
- **Between Limit 1 and 2**: If the robot is between these two limits it starts to move slower.
- **Further than Limit 2**: If the robot is outside limit 2 it accelerates towards the maximum speed limit.

The flowchart over the program for this model can be seen in Figure 6.
Case 1: Both sensors in position $\alpha$ gives a distance shorter then limit 1

Case 2: If the left sensor in position $\alpha$ gives a distance shorter than limit 1 but not the right sensor in position $\alpha$

Case 3: If the right sensor in position $\alpha$ gives a distance shorter then limit 1 but not the left sensor in position $\alpha$

Case 4: If both sensors in position $\alpha$ gives a distance longer then limit 2

Case 5: If both or one of the sensors in position $\alpha$ gives a distance between limit 1 and limit 2

Case 6: If the right sensor in position $\beta$ gives a distance shorter then limit 1 but not the left sensor in position $\beta$

Case 7: If the left sensor in position $\beta$ gives a distance shorter then limit 1 but not the right sensor in position $\beta$

Case 8: Both sensors in position $\beta$ gives a distance shorter than limit 1
3.3 Implementation with Ultrasonic Sensors and IR sensor

This model has the same approach as the model II, described in section 3.2.2. The difference is that this one has an IR-sensor in the middle that always looks forward and the program looks on that sensor first. If there is an obstacle in the way an appropriate action is taken. This model was also optimized by modifying some minor methods and parameters, e.g. we implemented a method that checks the sensors over and over again until we have a value that is at maximum 3 centimeter from the last value.

3.4 Implementation with IR Sensors

The three models with IR sensors are based on the potential field method for obstacle avoidance. In the models the forces are calculated as described below in this section. The difference between the models are how the further direction is controlled, described in section 3.3.1, 3.3.2 and 3.3.3.

When the robot is moving forward the sensors measures the distance to an obstacle at some specific angles $\theta_i$. A set of imaginary force vectors are calculated by using the measured direction and the distance to the obstacles. The magnitudes of the vectors is defined as

$$F_{left} = \frac{d_{optimal}^2}{d_{measured}^2}$$

$$F_{right} = \frac{d_{optimal}^2}{d_{measured}^2}$$

The forces $\vec{F}_{left}$ and $\vec{F}_{right}$ can be expressed for each angle $\theta_i$ in Cartesian coordinates, as can be seen in Figure 7.

![Figure 7: The imaginary force vectors with angle $\theta$](image)
The reference system is set to be fixed on the robot, as can be seen in Figure 8. The y-axis goes through the center of the front wheels and the x-axis goes through the center of the sensors.

![Reference system for imaginary force vectors](image)

**Figure 8:** The reference system for our imaginary force vectors.

This resulting in

\[
\vec{F}_y = F_{left} \cdot \sin(\theta_i) \hat{y} + F_{right} \cdot \sin(-\theta_i) \hat{y}
\]

\[
\vec{F}_x = F_{left} \cdot \cos(\theta_i) \hat{x} + F_{right} \cdot \cos(\theta_i) \hat{x}
\]

A driven force \( \vec{F}_f \) is introduced. This force points in the direction of the robot, \( \hat{x} \). To calculate the resulting force the superposition principle is used for all vectors which gives

\[
\vec{F}_{tot} = \vec{F}_f - \sum_{\theta_i} \vec{F}_y + \vec{F}_x
\]

This vector points in the direction of the most suitable direction to move for avoiding collision. The magnitude of \( \vec{F}_{tot} \) gives the further speed of the robot.

### 3.4.1 Model I

In model I the robot should rotate \( \alpha \) radians to get to the further direction, this means that the coordinate system should rotate \( \alpha \) radians, as can be seen in Figure 9.
To rotate the coordinate system $\alpha$ radians, the position of the center of the wheels is moved a distance $L$. The distance $L$ can be calculated by using the equation of the circumsphere of a circle given as

$$L = 2\pi R \frac{\alpha}{2\pi} = R\alpha$$

The distance $L$ is calculated for the wheels with the same method

$$L = r\theta$$

where $r$ is the radius of the wheels and $\theta$ the angle the wheels should turn. The distances $L$ should be equal to each other which gives

$$\theta = \frac{R}{r}\alpha$$

telling how much the wheels should rotate to turn the coordinate system $\alpha$ radians. A dynamical model over the robot can be seen in Figure 10.

Figure 9: Shows the angle $\alpha$ to turn the robot

Figure 10: Model over the robots wheels
3.4.2 Model II

This model works similarly to model I with IR sensors, but unlike model I this model turns and moves forward at the same time. This is done through down scaling of the speed on one of the wheels. If the robot is supposed to turn left the left wheel is then slowed down linearly depending on the angle of the resulting force. The motor is at complete stop if the angle of the resulting force is $\frac{\pi}{2}$ radians, and going at the same speed as the other wheel if the angle is 0 radians.

Since this model only has static sensors, it can collect distance readings a lot faster than model I described in section 3.4.1, about 10 times as fast. Which means that it can move faster than model I.

A filter was implemented in this model to estimate the speed of the wheels. The filter is based on a finite difference method with a second degree of accuracy. The approximation used is

$$f'(x_t) = \frac{3}{2} f(x_t) - 2 f(x_{t-1}) + \frac{1}{2} f(x_{t-2})$$

This can be rewritten in the form $\dot{x} = Ax + Bu$ and $y = Cx + Du$. By using this approximation the system becomes

$$A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} -2 & \frac{1}{2} \end{bmatrix}, \quad D = \frac{3}{2}$$

The approximated speed was averaged over the last ten timesteps and if the speed was low enough the robot would turn around.

3.5 Programming issues and recommendations

All our code was written in C using nxtOSEK. We had experience with nxtOSEK and C before and knew that it worked well. We wrote our code in Notepad++ since it’s simple and works well for all languages.

One problem we had was with the IR-sensors. When we used Ultrasonic sensors we could read a value from one sensor and then directly after we could read a value from the second sensor. When we did so with IR-sensors we received the same value from both sensors. This problem was solved by adding a small wait between the two reads.
Another problem that we were not able to solve was a problem with reads with the IR-sensors. For example, when we read a value from one sensor and then tried to read from another sensor we received an old value to that sensor. We made a preliminary solution by reading twice. Or three times if we had three IR-sensors.

The problem with nxtOSEK is that it has a limited API. If for example lejOS were used, with Java, we would be able to do a lot more with the robots. It has more useful commands. Although, since we had used nxtOSEK, this was a good choice for us because we could get started making the code almost immediately.

4 Result

The result from the implementations presented in section 3 is described below. A film of how the final models worked can be seen at: http://www.youtube.com/watch?v=FDUS-0GFLi8&feature=youtu.be

4.1 Result with Ultrasonic Sensors

The result from the implementations with ultrasonic sensors presented in section 3.2 is described below.

4.1.1 Model I

The implementation is working, the robot avoided collection of objects well. It can be noted that there is space for improvement in the reaction ability and speed control of the robot.

4.1.2 Model II

The implementation is working almost in the same way as Model I described in section 4.1.1 above. The difference between model I and model II is the speed control. Model II has two different distance limits and uses these to determine the current speed. When the robot is far away from an obstacle the robot accelerates to the highest allowed speed, and does the opposite when it is close to an obstacle. In general, model II is performing the task with the same result as model I apart from the speed control.
4.2 Result with Ultrasonic Sensors and IR Sensors

The model works really well, it doesn’t collide with any objects, doesn’t get stuck but it still have the problem to find a way between two obstacles if they are too close together.

Sometimes the robot makes weird choices, for example it might turn left when there are obstacles to right much further away. But in overall the model is much better than the same model without the IR sensor.

4.3 Result with IR Sensors

The result from the implementations with IR sensors presented in section 3.4 is described below.

4.3.1 Model I

This model worked quite well in regards of getting away from obstacles, but it couldn’t move so fast. This is because the robot has to make a full sweep with its sensors, which could take up to 1.5 seconds depending on how accurate the sensor readings should be.

In a bad case the robot could be close to the wall when reading the front distance, and it has to read the other distances as well before it could update its direction. During the time it takes to read the other distances it would have hit the wall if it were moving to fast.

4.3.2 Model II

This implementation worked really well but with a small problem. The robot could on rare occasions collide with a corner of an object. This is because the front sensor could not see the object while one side sensor could. Other than this it avoided every object except very thin objects, like a table leg.
5 Conclusions

Overall our five developed models worked well, but some problems occurred, which is discussed below.

The ultrasonic sensors gave unreasonable distance values sometimes. A reason for this could be that they affect each other in a bad way when they are used in the same environment. The transmitted sound waves from the sensors reflect in different ways which could lead to that both sensors receive a sum of sound waves from both sensors. To avoid this misreading of the real distance we had to keep the distance to close objects large.

Since the ultrasonic sensors measures wrong sometimes, we implemented a method in the combined IR and Ultrasonic model, as described in section 3.3. This method measures each sensor over and over again until the error between the measured and the last measured value is less than some error parameter. After implementing this the robots worked better.

Another problem that occurred with the ultrasonic sensors was that it sometimes failed to see objects, this could be due to that the ultrasonic sensors only sends waves straight forward and collects the reflection in the same direction. If the surface that the wave collides with is perpendicular the wave, it is not reflected straight back but instead in another direction. This reflection is not collected by the sensor and the obstacle is not observed.

In our ultrasonic models we placed the sensors on top of the robot and with this construction the robot was not able to detect short objects. We adjusted this in our IR model to be able to detect short objects as well.

For the IR model we decided to implement the potential field method, this because it was the most suitable method of the methods described in section 2.2. This with respect to our knowledge and available equipment.

A problem with the potential field method is that the robot may get trapped. This occur when the robot runs into a dead end, like a U-shaped obstacle. The problem was solved by using an algorithm that identified that the robot was trapped and turned the robot around.

In all of our models the robot is moving and measuring distance data at the same time. Both the IR and ultrasonic sensors measured wrong distance data sometimes when they were rotating, this could in the worst case lead to a collision. To obtain a more accurate distance data we made a robot with stationary sensors pointing in three different directions.
Moreover, rotational sensors collect distance data slower than stationary sensors, since they have to rotate to a specific angle and then collect data. This leads to that the model with stationary sensors could make faster decisions and thereby turn in time and avoid collisions.

In the non-stationary sensors models the measuring takes about 1.5 seconds and during this time the robot moves forward and this may lead to a collision. In the stationary sensor model the measuring takes about 200 milliseconds which makes it possible to make faster direction decision than in the non-stationary model. Because of this, we can increase the robots speed and still guarantee collision avoidance.

In our stationary model that is described in section 4.3.2, we used three stationary sensors. The result was pretty good, but we couldn’t see thin objects like a table leg. This happened because the sensors could only see objects right in front of them and not in the wide area between the sensors. A solution to this is to increase the number of sensors pointing in different directions. The NXT brick only has four sensor ports which limit us.

Moreover, the IR sensors are a bit better for this task then the ultrasonic sensors. The IR sensors is better because they can observe all obstacles in front of the sensors, not just those that is perpendicular to the sensor as the ultrasonic sensor.

Thus, our final conclusion is that the IR model with stationary sensors worked best, we believe this is due to that the sensors are stationary and can measure data frequent and also that the measuring error is much smaller with stationary sensors.
6 References


