Pathfinding Robots Incorporating Belief Revision to Control Behaviours

# Overview

The objective of this research was to determine the feasibility of using belief revision to control the behaviour of agents in a real-world situation. To conclude this, we used mBot robots with multiple behaviours, (scout, roam, patrol and defend), that were placed on a black and white grid. The robots navigated the grid using various sensors with their movement being dictated by their current behaviour state. Belief revision was used to alter the current belief state which is then imposed on the robots through the use of a whistle.

# Setup and Materials Used

## Materials

The various materials required to perform this research are:

1. Two, or more, Makeblock mBot robots with the following sensors:
   * 2 line reader sensors
   * 1 ultrasonic sensor, and;
   * 1 audio sensor
2. A printed grid to be used as a map
3. A whistle to be used as the belief revision software output
4. Three programs split up by their specific usages
   * A C script named “RobotHardwareHandler.c” which is used to interpret the various sensors and control the hardware on the robot, (movement, sounds emitted, etc.)
   * A C++ library named “RobotNavigation.cpp” and it’s header which are used to help with navigation and determine routes based on the robot’s current behaviour state
   * Belief Revision software suite written in C++ named “RobotBeliefRevision.exe” that works by receiving data on the robot’s current operations and then affecting a change in one or more robot’s behaviour state based on a desired solution.

## Setup

The environment used during this research is quite simple, with a few unique requirements that are easily replicated. The instructions are as follows:

1. First, the robot must have the RobotHardwareHandler and RobotNavigation library installed on the robot’s memory; this can be done through the Arduino software suite. (**Note:** if you wish to manually tell the robot the size of the grid rather than allowing it to dynamically map it then you must change the variables GRID\_COLUMNS and GRID\_ROWS at the top of the script. Also, this is where you set the location of the red, blue, yellow and green ‘bases’ (titled RED\_BASE, BLUE\_BASE, etc.). These should be coloured on the grid so viewers can tell where they are.)
2. Secondly, you should turn on the robots and place them close to the grid. Once this has been done, you can initiate the RobotBeliefRevision program and pair it via wireless with the robots you are using. This is essential for the belief revision software to work or else it won’t have an accurate idea on what state the robots are currently in.
3. Third, you should take the robots, one at a time, and put them on an outer corner of the grid (but not directly on the corner), facing clockwise, with their inner left sensor above the black line. After you place a robot down you must simply press the button on its casing and it will start driving and mapping the grid. Once it’s gone a fair distance from the corner you can safely place another robot there. (**Note:** all robots must be placed on the same corner when they are started up. This is how they determine their position in the grid as they are driving around.)
4. Finally, using the belief revision software you can create a set of beliefs for the robots to follow. When creating a belief state, the software will give you the corresponding amount of whistles you need to perform in order to change the robots’ beliefs to those aligned with the belief state you wish to enforce. The various combination of states are achieved using two whistle lengths and a combination of up to 5 whistle blows. Once a whistle is blown, the robots will signal that they are listening by ceasing movement and will consider the signal finished when there is no whistle sounds for 1.5 seconds or more. The whistle lengths used are:
   1. Short whistle burst; from 0.1-0.5 seconds in length.
   2. Long whistle burst; 0.5 or greater seconds in length.

# Hardware and Limitations

For this project, we used Makeblock mBot robots and various add-on components from Makeblock. Running on the ATmega328 chip with only 32KB of programmable flash memory and 2KB of RAM, the mBot has some restrictions due to its limited processing power. We had to separate the belief revision software, run it on an external computer, and then interface with the mBot robot via 2.4GHz wireless communication to circumvent the mBot’s limited power. The primary components used, aside from the on-board 2.4GHz wireless transiever, are two line-reader components, one ultrasonic component, and one sound sensor component.

The line-reader components are used to navigate through the grid, and do so through the use of two IR emitting LEDs and two IR sensitive phototransistors on each. Each LED shines a beam onto the ground underneath it and if that beam hits a surface that is reflective (like white paper) then the beam will be reflect back onto the phototransistor. Likewise, if the IR beam is absorbed by the surface being shined on (like black paper) then no light will return to the phototransistor. Through this process the robot is able to determine what part of a grid it is driving on and use that information for navigation. Although these sensors are highly accurate, there can be an issue if the material underneath the line-reader components is somewhere between matte and reflective.

The sound sensor on each robot is used to receive instructions from the belief revision software. After creating a new belief state, the software will tell the user how many times they should blow a whistle, and for how long, which will then be interpreted by the robots. Upon hearing a whistle, the robots will stop movement and listen, and when the whistle sequence stops they will interpret that as an instruction which alters their current behaviour state. The sound sensor works when the whistle sound is loud enough but, due to being mounted on a robot with loud motors constantly running, it is difficult for the robot to process a new sound coming in if it is too quiet.

The final primary component, the ultrasonic transmitter and receiver duo, is used to prevent collisions with other robots while navigating the grid. It works by emitting ultrasonic sounds in front of it, and is then able to detect the sounds if they bounce off of an object and back to the ultrasonic detector. Based on the length of time that’s elapsed from when it sent the sound and when it received it back, it is able to tell how far away an object is. The ultrasonic component works well if used to detect the range of a large, static object like a wall but has issues with small, moving objects like other robots. The robots are programmed to move at slower speeds to combat this deficiency.

# Software Used

The robot hardware handler script is written in C through the Arduino IDE and is built to handle the various hardware and components on the robot. Among other things, this script stores the current state of the robot, (scout, roam, patrol, or defend), and its current position. Although it handles the operation of the motors, speakers and wireless transmitters, the majority of the code handles the robot’s line-reader sensors and interpreting what the robot is seeing. The script will take in raw data from the robot’s line-readers and, based on the robot’s current and previous states and position, will make a determination on what corrective actions are needed. For example, if the robot’s leftmost sensor is reading black and the other three sensors to the right of it are reading white then we know the robot is slightly off-centre of the grid line that it is over top of.  If the current state of the robot was to ‘drive straight’ then the robot would know that it needs to correct itself and it would drive left until the middle sensors were over the grid line, then it would continue to drive straight.

When the hardware handler script makes the determination that something has changed with the robot, (i.e. the line-readers have indicated that the robot is now at a crossroads in the grid), the script will then ask the robot navigation library what its next action should be. This library has functions that take in the robot’s current state and position, calculate a route for the robot, and then return a direction for the robot to go in. For example, if the robot’s current state is to defend position [1, 1] of the grid then the navigation library will create a path to that point in the grid. If the robot reached an intersection at point [1, 0] coming from [0, 0] then the navigation library would tell it to turn right and then go straight so it could reach [1, 1]. Also, if it were to run into another robot on the way to that point, it would then determine an alternate route to get to position [1, 1] that circumvents the intervening robot.

Since we start all the robots in the same spot on the grid, they all have the same concept of what the grid looks like. This assumption makes it a lot easier for us to send belief revision commands to the robots as we can include a grid position in the commands, like [2, 1], and not worry that there will be a discrepancy between each robot on where that is. The robot will only ask the library what decision to make when it meets a crossroads in the grid, detects another robot with its ultrasonic sensors, or receives orders from the belief revision software.

The belief revision software is used when wanting to change the current belief state. The program is still undergoing work and the end result should output a signal that is interpreted differently by each robot with the outcome of all robots adjusting their behaviour states to match the new belief state. Currently, whistle blows are used to signal the robots that a change of belief state has occured. With 0-16 short or long whistle bursts, we are able to adjust the robot’s behaviours to fit our desired belief state; this means that with one command all the robots on the grid can take on new behaviour states.

# References

ATmega328 Chip Documentation: http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P\_datasheet\_Complete.pdf