BUILDING INTELLIGENT AGENTS AS AN ITERATIVE PROCESS OF INCREMENTAL SENSING ABILITY

Riccardo Luca Broggi
Department of Computer Science
University of Bath
rlb47@bath.ac.uk

Bdour Al Lawzi
Department of Computer Science
University of Bath
bal28@bath.ac.uk

February 28, 2019

1 Introduction

This paper describes the process of building an increasingly intelligent agent by iteratively enhancing its capacity to sense the world and cope with the problem of action selection by means of richer sensory input. The main motivation for this research is the insight that building an agent by decomposing its intelligence into parallel and independent action-producing behaviours, would lead to greater success in proportion to its ability to sense the world Brooks (1991). This reactive approach to building intelligence does away with the need of representations and relies instead on interfacing with the world through perception and sensing. The hypothesis was that an agent's intelligence can be improved by adding layers of independent behaviour in a hierarchy, incrementing its sensory ability to perceive its environment; a robot following a wall and navigating in the environment, which was chosen as a proxy of the research. It was found that the agent was indeed displaying increasingly intelligent behaviour as its ability to sense the world increased, however some concerns with this approach arem also raised and discussed.

2 Approach

To be able to experiment and test the hypothesis, an agent was built using a LEGO EV3 Robot and the Lejos Behaviour library, enabling the creation of a subsumption architecture. This allowed the problem of building an incrementally more intelligent agent to be sliced into a set of parallel, task-achieving sub-behaviours.

The Subsumption library used enables this architecture by defining a set of behaviours that are ordered by priority, whose active/suppresed state is established based on sensory information. This Object Oriented Design paradigm is coherent with the argument that "on execution a competence checks the trigger of each of its elements to find the highest priority element that can run; [...] the environment is always re-sampled for selecting the next element" (Bryson & McGonigle 1998).

The first step was therefore to abstract the problem from most details to form a simple description in terms of atomic concepts encapsulating the environment, such as ROBOT, WALL, OBSTACLE. It was then decomposed on the basis of external manifestations and behaviours, defined as levels of competence in Brooks (1986):1) ROBOT move around, 2) follow WALL moving, 3) avoid contact with OBSTACLES.

Level 0 competence was comprised of a single behaviour requiring no sensory input, instructing the robot to move forward without any feedback from the environment.

Level 1 enables the robot to follow the wall tackling the issue of localisation, which is essential to navigation. Nirmala et al. (2017) argue that Landmark Navigation is an effective system to achieve this, and therefore a second iteration of the robot was built with an ultra-sonic sensor aimed at a wall, using this sensory input to navigate the environment by keeping a constant distance from the wall. Anytime the distance from the wall falls outside a defined range, a corrective measure is taken by the robot: initially a fixed standard corrective measure was taken. However, experimentation revealed a correction proportional to the displacement from the range; leading to a more accurate wall following and fewer collisions but with a longer circuit time.

To further increase the level of intelligence of the agent and take on competence level 2, the concept of autonomy had to be entertained: as experimentation showed, simply following a wall lead the robot to getting stuck whenever the morphology of the environment presented obstacles that were not within the range of sensing of the ultra-sonic sensor. A touch sensor attached to a front bumper was therefore introduced, adding an extra sensory input for the robot to navigate the environment and detect when an obstacle was hit.

As will be argued in the results section, this lead to a substantial improvement in the intelligence of the agent as it required far fewer interventions, however some limitations were observed with the recovery time. To further improve its decision making, a second touch sensor was introduced attached to the same bumper on the front of the robot.

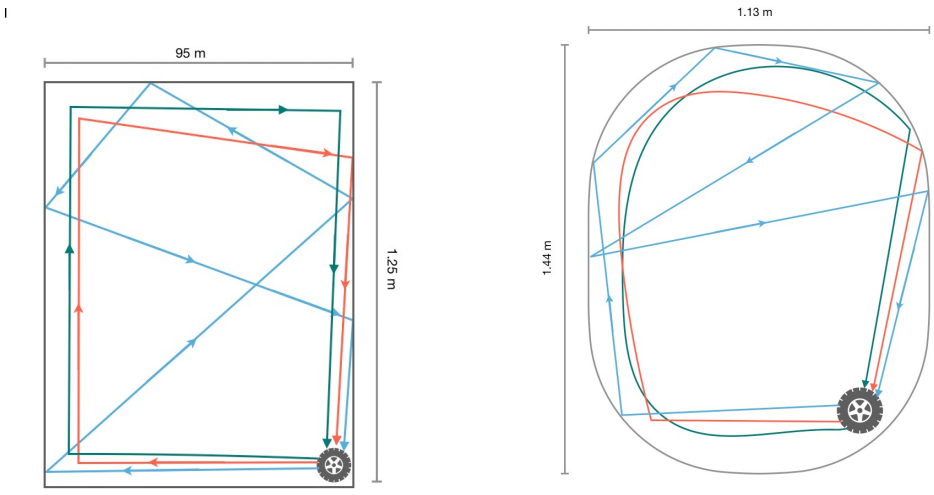
This decision was based on Brooks (1991) who argued that "activity producers all interface directly to the world through perception and action"; the emphasis here is on activity production based on perception which is different from mere sensation, in that perception relies on comparing sensation with respect to a set of expectations.

In this case, the agent is able to distinguish the position of the obstacle via the dual-dimensional information from the two touch sensors and its expectation of where it might be in light of the orientation of the ultra-sonic.

Throughout the process, improvements were achieved by means of altering the morphology of the robot, altering for instance the shape of the bumper to cover a larger area - by means of wider L shaped bars as well as low-hanging bars to detect short obstacles otherwise going undetected - and the orientation of the ultra-sonic sensor from horizontal to vertical to get a surface perpendicular reading and cope with corners better. Further, optimisation in the values used for range and angle of the sensors were made based on experimentation to improve the behaviour of the agent, however it was not strictly relevant for the hypothesis being tested.

3 Results

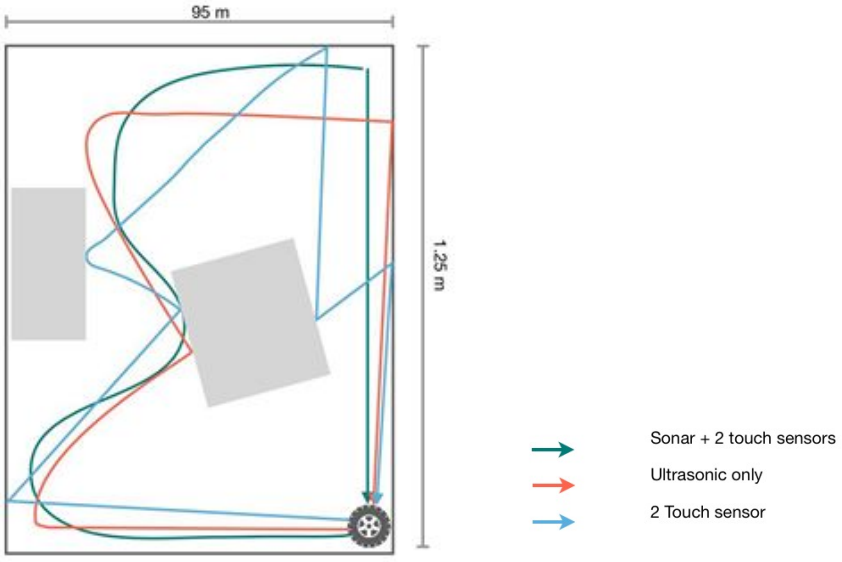
To test the hypothesis, experiments in 3 conditions were conducted for every configuration of the robot across several trials to witness average behaviour: a rectangular-shaped environment (figure 1a), circular-shaped environment (figure 1b), and an environment with uneven walls and obstacles (figure 2a). Figures 1a-2a show the typical behaviour of the robot in these conditions



(a) Environment 1 - Rectangle.

(b) Environment 2 - Circular.

Figure 1: Part 1 - Typical behaviour of different environments at different layers of sensor abilities.



(a) Environment 3 - Uneven wall and obstacles. (b) Environment Sensory Abilities key.

Figure 2: Part 2 - Typical behaviour of different environments at different layers of sensor abilities.

4 metrics were observed that capture the intelligence gained by adding sensory abilities incrementally: average time taken to complete a circuit, the number of collisions, average recovery time from collision, and the number of human interference in a circuit.

Each sensory ability was run 10 times, in each of the 3 conditions and averaged to obtain the results below.

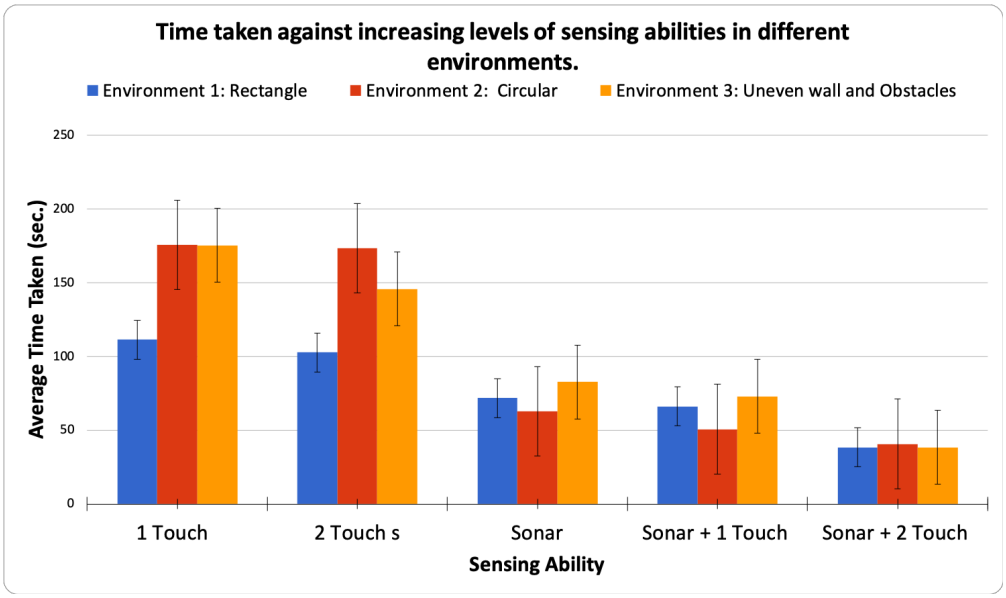


Figure 3: Time taken against increasing levels of sensing abilities in different environments.

Figure 3 shows that the robot's intelligence was incrementally developed by the implementation of additional sensory inputs, resulting in the decrease of the time taken to complete a circuit. Across the different environments similar behaviours were witnessed, as well as seeing longer time in the third environment, due to collisions with present obstacles. Further metrics will be examined to explore the behaviours in more detail.

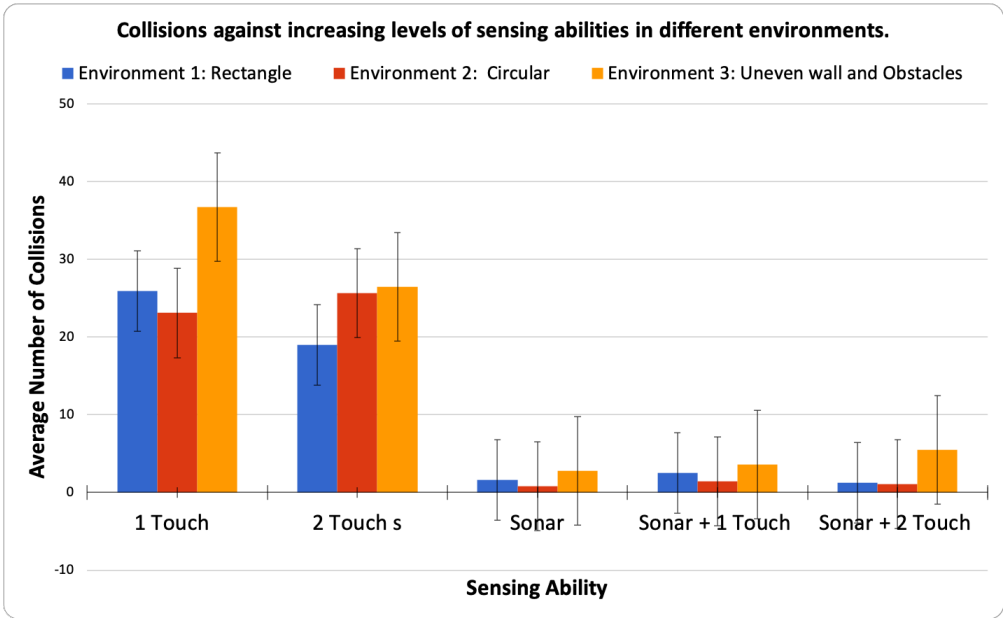


Figure 4: Collision against increasing levels of sensing abilities in different environments.

Figure 4 shows that the number of collisions drastically decreases when the ultra-sonic sensor ability is added, as it addresses the issue of localisation such that the robot is able to navigate in its environment. As expected, in the environment with more obstacles there are more collisions. There is a large number of collisions when only the touch sensor ability is present; as the robot do not have an up-to-date view of the environment. However, there are less collisions observed when a secondary touch sensor is added; giving it the ability to make better informed decisions.

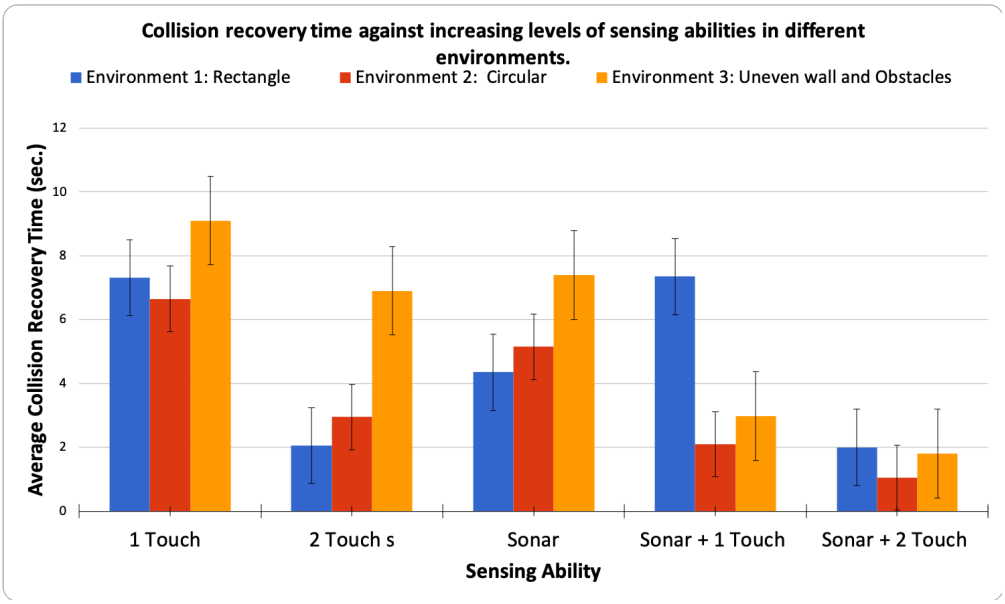


Figure 5: Collision recovery time against increasing levels of sensing abilities in different environments.

In figure 5, the same trend can be seen; due to the presence of more sensory input, the recovery time from a collision decreases. One observation from this experiment, which was stated in the approach, was that having one touch sensor requires more recovery time; attributed to the lower sensitivity and stability of attaching a single touch sensor. This was evident as well even when the robot had ultra-sonic and 1 touch sensor ability. As an informed decisions on the rotation to avoid the obstacle could not be made, the robot needs time to get back in the range to follow the wall.

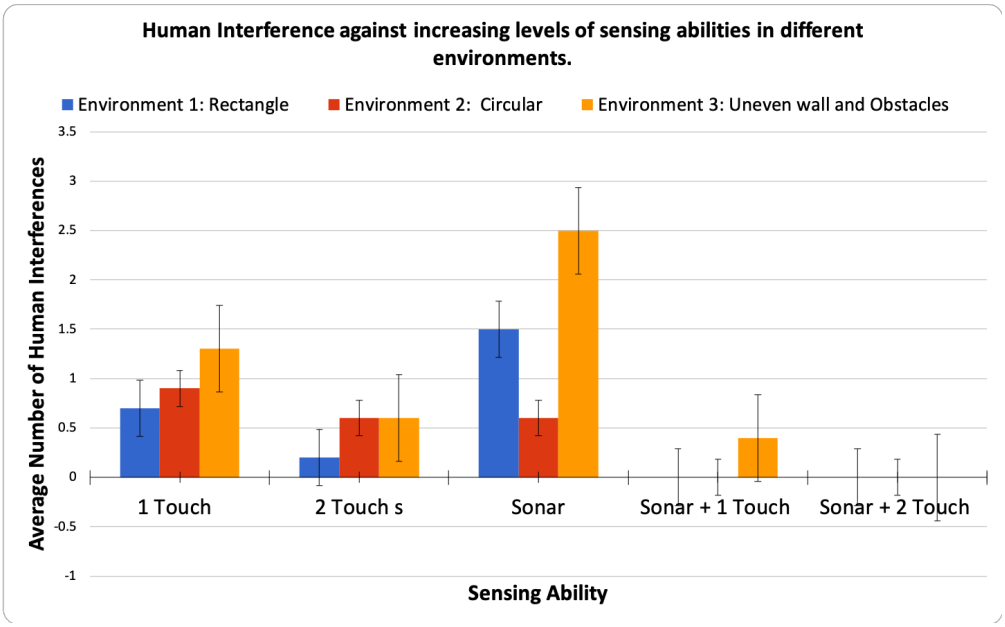


Figure 6: Human interference against increasing levels of sensing abilities in different environments.

The figure 6 metric addresses the intelligence in terms of a robot being able to cope for itself without human interference. As more sensory input was incremented, less if any human interference is needed.

The collection of the metrics supports the hypothesis; incrementally enabling task-based behaviour, by combining the layers of sensory abilities in-parallel, allows the robot to make better informed decisions.

4 Discussion

Although, the results presented above seem promising, there is room for concern among the authors that the success achieved in this research was by virtue of the large marginal improvements available when going from no intelligence to very limited beginner intelligence. The speculation is that these "low-hanging fruits" are limited, and that scaling this approach to reach true intelligence would hit a limit. The limitations of the sumbsumption architecture is further expressed by (Brooks 1991), who discusses the complexity of coordination between behaviours.

Another reason for concern is the purely Markovian approach of basing behaviour merely on sensory input, thus assuming the action production problem does not need past sensory input. Observation from the experiments suggests this may not be true; in certain circumstances the robot would get stuck in a loop and require external intervention. To solve this, the author speculate the subsumption architecture could still accommodate a "memory" behaviour, however more research would need to be carried out.

Finally, judging the degree of intelligence of an agent is a hard task, and presumes an agreed upon definition; it was decided to use the notion of "doing the right thing at the right time" and considering the aim of intelligence "to get better at doing the right thing at the right time". In light of this, further work would focus on introducing behaviour aimed at implementing reinforcement learning techniques. This would enable greater intelligence, such as exploring around the environment and learning how best to navigate it.

5 Conclusion

This paper has presented the experimental findings, supporting our hypothesis, of building an increasingly intelligent agent by means of adding parallel behaviours in a subsumption architecture leveraging increasing availability of sensory information. This approach to incremental intelligence development via parallel behaviours has been shown to be successful and seems promising. However, more research needs to be conducted into the limitations of a purely Markovian reliance on sensory input. An interesting next research will be on how to add memory as a means of incorporating learnings of priors, biases and expectations.

References

Brooks, R. (1986), 'A robust layered control system for a mobile robot', *IEEE Journal on Robotics and Automation* **2**(1), 14–23.

Brooks, R. A. (1991), 'Intelligence without representation', *Artificial Intelligence* **47**(1), 139 – 159.
URL: *http://www.sciencedirect.com/science/article/pii/000437029190053M*

Bryson, J. & McGonigle, B. (1998), Agent architecture as object oriented design, *in* M. P. Singh, A. Rao & M. J. Wooldridge, eds, 'Intelligent Agents IV Agent Theories, Architectures, and Languages', Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 15–29.

Nirmala, G., Geetha, D. S. & Selvakumar, D. S. (2017), 'Mobile Robot Localization and Navigation in Artificial Intelligence: Survey', *Computational Methods in Social Sciences* **4**(2), 12–22.
URL: *https://doaj.org*

6 Appendix

6.1 Code

File: ICCSRunner.java

```
import lejos.hardware.*;
import lejos.hardware.port.SensorPort;
import lejos.hardware.sensor.EV3TouchSensor;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.subsumption.*;


public class ICCSRunner
{

    public static void main(String[] args) {

        System.out.println("Press␣to␣start");

        Button.waitForAnyPress();

        Behavior drive = new DriveForward();
        Behavior detectHit = new DetectHit(new EV3TouchSensor(SensorPort.S3), new EV3Touch
        Behavior follow = new FollowWall(new EV3UltrasonicSensor(SensorPort.S4));


        Behavior[] bs = {drive, follow, detectHit};
        Arbitrator arbitrator = new Arbitrator(bs);
        arbitrator.go();

    }
}
```

File: DriveForward.java lstinputlisting[language=Java]Code/DriveForward.java

File: FollowWall.java lstinputlisting[language=Java]Code/CodeFollowWall.java

File: FollowWallProportional.java

```
import lejos.hardware.motor.Motor;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.subsumption.Behavior;

public class FollowWallProportional implements Behavior
{
        private EV3UltrasonicSensor     ultrasonicSensor;
        private boolean suppressed      = false;
         private float[] sample;

        public FollowWallProportional(final EV3UltrasonicSensor ultrasonicSensor)
        {
                this.ultrasonicSensor = ultrasonicSensor;
                sample = new float[1];
        }

        @Override
        public boolean takeControl()
        {
                        ultrasonicSensor.getDistanceMode().fetchSample(sample, 0);
                System.out.println( "S:" + sample[0]);
                return !(Math.round(sample[0] * 100.00)/100.00 < 0.2 && Math.round(sample[0
        }

        @Override
        public void suppress()
        {
                suppressed = true;
        }

        @Override
        public void action()
        {
                suppressed = false;
                 Motor.A.setSpeed(600);
                 Motor.B.setSpeed(600);
                 int proportionalCorrection;

                 if(sample[0] < 0.16) {
                        proportionalCorrection = (int) (( 0.16 - sample[0] ) * 100);
                        if(proportionalCorrection > 20) proportionalCorrection = 20;
                        if (proportionalCorrection < 5) proportionalCorrection = 5;
                Motor.A.rotate(proportionalCorrection, true);
                //System.out.println("sample "+ sample[0] + "rotate" + proportionalCorrecti

                } else if(sample[0] > 0.21) {
                        proportionalCorrection = (int) (( sample[0] - 0.2) * 100);
                        if(proportionalCorrection > 20) proportionalCorrection = 20;
                        if (proportionalCorrection < 5) proportionalCorrection = 5;
                Motor.B.rotate(proportionalCorrection, true);
                //System.out.println("sample "+ sample[0] + "rotate" + proportionalCorrecti
        }

                while(!suppressed && Motor.B.isMoving()) Thread.yield();
        }

}
```

File: DetectHit.java

```
import lejos.hardware.motor.Motor;
import lejos.hardware.sensor.EV3TouchSensor;
import lejos.robotics.subsumption.Behavior;

public class DetectHit implements Behavior {

        private boolean suppressed = false;
        private EV3TouchSensor rightBumper;
        private EV3TouchSensor leftBumper;
        private float[] sampleRight;
        private float[] sampleLeft;

    public DetectHit(EV3TouchSensor rightBumper, EV3TouchSensor leftBumper){
        this.rightBumper = rightBumper;
        this.leftBumper = leftBumper;
        sampleRight = new float[1];
        sampleLeft = new float[1];
    }

    public boolean takeControl() {
        rightBumper.getTouchMode().fetchSample(sampleRight, 0);
        leftBumper.getTouchMode().fetchSample(sampleLeft, 0);
        return sampleRight[0] == 1 || sampleLeft[0] == 1 ;
    }

    public void suppress() {
       suppressed = true;
    }

    public void action() {
                Motor.A.setSpeed(200);
                Motor.B.setSpeed(200);
        suppressed = false;
        Motor.A.rotate(-360, true);
        Motor.B.rotate(-360, true);
        if(sampleRight[0] == 1 && sampleLeft[0] == 1) {
                Motor.B.rotate(90, true);
        } else if(sampleRight[0] == 1) {
                Motor.B.rotate(90, true);
        } else if(sampleLeft[0] == 1) {
                Motor.A.rotate(90, true);
        } else {
                System.out.println("DIDNT CONSIDER THIS!!");
        }
        while(!suppressed && Motor.B.isMoving()) Thread.yield();
    }


}
```