#define DEFAULT\_SPEED 50

#define SONAR\_ADJUST 18

#define SONAR\_THRESH 100

int sonarDistance = 999;

void stopMotors() {

motor[motorA] = 0;

motor[motorB] = 0;

}

// moveBackwards moves the robot backwards for a given amount of time.

// int power the power of the motors

// int time the amount of time the motors will run.

void moveBackwards(int power, int time) {

motor[motorA] = -1 \* power;

motor[motorB] = -1 \* power;

sleep(time);

}

// turnLeft turns the robot left for a given amount of time.

void turnLeft(int power, int time) {

motor[motorA] = -1 \* power;

motor[motorB] = 1 \* power;

sleep(time);

}

// turnRight turns the robot right for a given amount of time.

void turnRight(int power, int time) {

motor[motorA] = 1 \* power;

motor[motorB] = -1 \* power;

sleep(time);

}

task attack()

{

while(1) {

if (sonarDistance > SONAR\_ADJUST && sonarDistance < SONAR\_THRESH)

{

motor[motorA] = 1.0 \* sonarDistance;

motor[motorB] = 1.0 \* sonarDistance;

} else if (sonarDistance < SONAR\_ADJUST)

{

stopMotors();

sleep(2000);

moveBackwards(DEFAULT\_SPEED, 1000);

if(random(1))

{

turnLeft(DEFAULT\_SPEED, 400);

} else

{

turnRight(DEFAULT\_SPEED, 400);

}

} else {

stopMotors();

}

}

}

/\*\*

Check, read and assign the values sent from the slave brick

\*/

void readBluetoothMessages() {

//Something got through if the signals are not all zero

if(messageParm[0] != 0 || messageParm[1] != 0 || messageParm[2] != 0)

{

sonarDistance = messageParm[0];

//fearSensor = messageParm[1];

//nxtDisplayBigTextLine(2, "S: %d", sonarDistance);

//Clear the messages, set them to 0

ClearMessage();

}

}

task main()

{

startTask(attack);

while(1)

{

readBluetoothMessages();

nxtDisplayBigTextLine(2, "S: %d", sonarDistance);

sleep(100);

}

}