# Driver.h

void decideLeftRight(int time, int speed=MOTOR\_SPEED){

Decides which direction it goes (Left or Right). This is randomized, but if there is an obstacle on the way, then it will avoid the obstacle. Once a random direction is selected, then Arduino will drive towards that direction for 300 ms (time) at 50 units of power from the battery (speed).

void decideForwardBackward(int time, int speed=MOTOR\_SPEED){

This function is only executed when the Arduino car is lost. 80% of the time (random number picked is between 1 and 4) the car will move forward. Then 20% of the time (when random number picked is 0), the car will move backwards.

boolean wasLost = false

boolean resetWasLost(){ wasLost = false; return false; }

Here we are defining a variable and a function. The variable is set to False as default. When wasLost is False, then the Arduino car is allowed to get lost. If wasLost is True, then we don’t let the car get lost again (else it gets lost a lot).

boolean iAmLost(){

if(wasLost || voiceControlled){

return false;

}

This function is run on a loop. If the car is lost, then return false (which does nothing). If there is somebody voice controlling the car, then return false (so that the car also never gets lost). However, if the Arduino robot was not lost, then the function continues:

// We are lost... Let's stop for a moment

drive(0);

// Look around and go to a random direction

decideLeftRight(300, 50);

drive(0);

delay(100); // talk here: e.g. “Help me, I am lost!”

decideLeftRight(300, 50);

drive(0);

delay(100); // talk again? E.g. “ I cannot find my way, help”

decideForwardBackward(300, 50);

// wait 1 minute before we can get lost again

wasLost = true;

timer.in(60000, resetWasLost);

return true;

}

The robot performs wandering behavior, randomly going left or right twice, then back or forward once. Within this time, the robot expresses confusion, asking the human for help and guidance. The function ends by setting the wasLost variable equal to True, so that the Robot no longer behaves confused for a full 1 minute. This makes the robot wander around with simple obstacle avoidance and no randomization of behavior (no intentional wandering around). The last part of the code is for simple driving with obstacle avoidance:

Text

Description automatically generateddir == 0 means the car is stuck

dir == 1 means the car move forward (no obstacle ahead, it can keep on going)

dir == 2 means the car can go to the left (no obstacle to the left)

dir == 3 means the car can go to the right (no obstacle to the right)

So this self-driving function says:

If there is an obstacle ahead (dir!=1) and if the car is stuck (dir==0), then drive backwards for 1000ms and to either the left or right for 500 ms.

If the car is presented with an obstacle directly in front of it (dir!=1), but it is not fully stuck (dir is not 0) then it drives to a direction (dir) where there is no obstacle for 500ms.

But as long as there is no obstacle ahead, then the car always keeps driving forward:

if(!iAmLost()) drive(1);

The IAmLost function was added to randomize this aspect, since it is not normal for a human to always walk around strictly forward until they face an obstacle. We usually wander around a bit, mostly moving forward, but still checking different directions from time to time.

# 

# Motor.h

Text

Description automatically generated

Here we define some global variables for the motor.

MOTOR\_SPEED\_SLOW is the speed we want to use for slower movements, e.g. while dancing.

MOTOR\_SPEED is the speed we use while driving normally

MOTOR\_STANDBY is the PIN that is used for the motor to completely turn the motor on or off.

MOTOR\_RIGHT is the PIN to move the wheels on the right.

MOTOR\_RIGHT\_DIR is the PIN we use to determine the direction the wheels turn.

MOTOR\_LEFT and MOTOR\_LEFT\_DIR are the same, but for the left side.

Text

Description automatically generated

This initializes the motor and is ran when the Arduino starts.

It tells the Arduino that all pins are used as OUTPUT, that means the Arduino can send power through these PINS to the motors.

Text

Description automatically generated

Here we defined a Boolean called “handbrake”. If handbrake = true, the main loop is not able to trigger the function for driving forward automatically using the SelfDriving() function. (See the “Selfdriving()” function explained in the Driver.h file).

stopMotor is a function that will stop the motor completely. The function takes in a parameter (brake) whose default is true. If instead the parameter brake is false, then the variable handbrake will be set to false, and the robot will keep on driving. Note: this stopMotor() function is useful for when we tell the robot to completely stop (brake=true) or when we want the car to go in a certain direction for a limited amount of time. For instance, when you tell the robot to go to the left, then it will keep on going to the left (wheels changing direction) until you tell the wheels to stop (break=false).

We stop the motor by writing “LOW” to the MOTOR\_STANDBY PIN (that means, no power to this pin)

We then set the speed of both the left and right wheels to 0, to make them stop turning.

Here we use “analogWrite” instead of “digitalWrite”.

digitalWrite allows a value of either HIGH (1) or LOW (0), analogWrite allows values between 0 and 255. This allows us to define a speed more precisely.

After all this we overwrite the “handbrake” variable with the passed parameter.

Text

Description automatically generated

This function starts the motor, it does the opposite of stopping by sending HIGH to the MOTOR\_STANDBY and passing the new speed to the MOTOR\_RIGHT and MOTOR\_LEFT.

This speed needs to be passed along through a parameter (speed);

Since we are now driving, we remove the handbrake to allow the Arduino to move forwards triggering the “Selfdriving()” function from the main loop. Note: the main loop is all the code inside the loop in the Arduino\_mega.ino main file

Text

Description automatically generated

This function sets both MOTOR\_LEFT\_DIR and MOTOR\_RIGHT\_DIR to HIGH so all the wheels will make the Arduino move forwards.

Text

Description automatically generated

This function sets both MOTOR\_LEFT\_DIR and MOTOR\_RIGHT\_DIR to LOW so all the wheels will make the Arduino move backwards. When the direction pins 7 and 8 have high voltage, they will make the wheels move forwards, and when the direction pins have low voltage, they will make the wheels move backwards.

Text

Description automatically generated with medium confidence

This function sets MOTOR\_LEFT\_DIR to LOW and MOTOR\_RIGHT\_DIR to HIGH. This way the **left wheels** moves backward, and the **right wheels** move forwards. This will make the Arduino turn to the **left**.

Text

Description automatically generated

This function set MOTOR\_LEFT\_DIR to HIGH and MOTOR\_RIGHT\_DIR to LOW. This way the **left wheels** moves forward, and the **right wheels** move backwards. This will make the Arduino turn to the **right**.

Text

Description automatically generated

This is the drive function we use in other places in our code to trigger all the functions explained above. We can pass in the direction to move (dir) for how long to move in this direction (timer, in ms) and with what speed we want to move (speed).

The direction can be either:

* -1 to go backward
* 0 to stop the motor (and put the handbrake on, handbrake = true)
* 1 to move forwards
* 2 to turn left
* 3 to turn right

The timer defaults to 0, this will imply that it will keep moving in that direction until this function is triggered again.

The speed is set to default to the MOTOR\_SPEED variable we defined at the top of this file.

If the dir==-1 we trigger the goBackward() function

If the dir==0 return the stopMotor() function, this triggers the function and stops the rest of this function from being executed. This prevents it from triggering the “startMotor()” function used later on.

If the dir==1 we trigger the goForward() function

If the dir==2 we trigger the goLeft() function

If the dir==3 we trigger the goRight() function

After these are triggered and the wheels are set to spin in the right direction, we start the motor based on the speed we got from the parameter “speed”.

If a timer was given, we use a delay to keep the wheels from turning in this direction until the time has passed, after this we trigger the stopMotor() function to stop turning in this direction, but we set the handbrake to false so the main loop can trigger the “selfDriving()”.  
This way the Arduino can keep on driving forward into a given direction for a while.

# ObstacleAvoidance.h

Text

Description automatically generated

This defines the global variables used for the ultrasonic module.

We set a MAX\_DISTANCE (30, in cm) which defines when we detect an obstacle.

We also set both the ULTRASONIC\_TRIGGER and ULTRASONIC\_ECHO, which are the PINS used for the module to communicate with the Arduino (12 for the trigger, 13 for the echo), more about this later.

Text

Description automatically generated with low confidence

Here we define the Servo, this is the tiny blue rectangle below the face of the Arduino.

This is a small motor that allows us to move the head left and right.

We use the Servo library for this (<Servo.h>) and initialize an **instance of the Servo class on a variable called useServo.**

We also set the PIN that the servo uses to communicate with the Arduino (10).

Text

Description automatically generated with medium confidence

This function is run when the Arduino starts and initializes the ultrasonic and Servo.

We set the ULTRASONIC\_TRIGGER as an OUTPUT PIN, this way we can send power over this PIN to the ultrasonic module so it can output a soundwave. We set the ULTRASONIC\_ECHO as an INPUT so the ultrasonic module can let us know when it got an echo back from the soundwave that was sent.

We also initialize the Servo using “useServo.attach()”, we pass the PIN number the Servo is using to communicate with the Arduino. After that is done, we trigger “useServo.write()”. The number we pass here is the amount of degrees the Servo needs to rotate the head, which is 90 to look forward.

Graphical user interface, text

Description automatically generated

We define a variable that allows us to determine which direction the head is currently looking.

* 0 means forward
* 1 means left
* 2 means right

After that we create a function that will make the head look forwards.

We can pass in a parameter (wait) to tell the Arduino to wait for the movement of the Servo to be done before continuing with the main loop. This is true by default.

If we are currently already looking forward (lookingInDirection==0), we stop the code (return), since there is nothing we have to do.

When this is not the case, we will trigger “useServo.write(90)” to make the head look forwards (90 degrees, as explained before).

If the wait parameter that we passed when using this function is true, we will wait 10ms for each degree the Servo has to move. This is always 90 (so 900ms) since we either were looking left (180 degrees) or right (0 degrees) before. So it is always 90 degrees from either of those locations. We need this delay, else the rest of the code would continue running before the robot is actually facing the right direction.

At the end of this function, we overwrite the “lookingInDirection” variable to the direction we are currently looking at (0, which is forward).

Graphical user interface, text, application

Description automatically generated

This essentially does the same as the previous function, but for looking left (direction 1).

The main difference is inside the if(wait) statement. When we were looking to the right before, we now have to turn the Servo twice as far (180 degrees) and thus we wait 1800ms (10ms per degrees) instead. If we were looking forward (direction 0) we only have to move 90 degrees to look to the left.

Text

Description automatically generated

This does the same as the “lookLeft()” function, but then for looking to the right (direction 2).

Graphical user interface, text, application, email

Description automatically generated

The “detectObstacle()” function is used to detect if there is an obstacle close by.

We write a LOW to the ULTRASONIC\_TRIGGER PIN to make sure it currently isn’t sending anything. We need to keep this running for 2 microseconds as followed by the example seen here:

<https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6#about-project>

After 2 microseconds we will write a HIGH to the ULTRASONIC\_TRIGGER PIN, this will make the Ultrasonic module output an ultrasonic soundwave. We then let it keep on sending soundwaves for 10 microseconds in total (also following the example as seen in the link shared above). After 10 microseconds passed, we can stop outputting soundwaves so we set the ULTRASONIC\_TRIGGER PIN to LOW again.

Now we can get the total duration in milliseconds from the ULTRASONIC\_ECHO PIN. We can get this value using the pulseIn() function to read the value, passing HIGH to tell the ultrasonic module to return the **duration it took to get back the echo** from the ultrasonic soundwave.

Now that we know the number of milliseconds it took to get a response, we can calculate the distance. The speed of sound is 3.4m/s, this is **0.034 centimeters per second**. The duration of the echo first travelled to the object and then back. Knowing this we can use the following calculation: distance (in cm) = duration (in ms) \* 0.034 / 2

Now that we have the distance in cm, we return “distance < MAX\_DISTANCE”

MAX\_DISTANCE was defined at the top of the file (30cm). So, **if the calculated distance is less then 30cm**, this function will return “true”, and we will know that an object is within a 30cm reach.

If the calculated distance is more then MAX\_DISTANCE (30) it will return “false”, and we will know that there is no object within a 30cm of the robot.

Graphical user interface, text, application, email

Description automatically generated

int avoidObstacles() pre-defines a function that is declared later in the code. C++ reads code from top to bottom, and if we don’t pre-define it here we are not able to use it inside of the randomlyDecideTimeout() function.

randomlyDecideTimeout() will be used with a timer (a library). Reminder: all the functions that are triggered by the timer need to return a boolean, to tell the timer whether it should or shouldn’t run the timer again (true / false).

The timer also enforces us to accept the void\* type as a parameter since it is possible to pass any type of variable along to the function we want to trigger with the timer.

randomlyDecideTimeout() is a function that we use when the robot got stuck while he is voice controlled. If it didn’t receive a voice command within 10s after it got stuck, the robot will decide a random direction on its own (as long as there is no obstacle there). **It is waiting for 10 seconds for the human to help him, and if he doesn’t get help then he proceeds driving on his own accord**

isStuckAndWaitingForCommand is a boolean declared in “voiceCommand.h” and is used to know if we just received a voice command. If we did (if we are not isStuckAndWaitingForCommand), we don’t want this function to continue since there is no reason for the Robot to pick its own direction now. So, we return false and tell the timer not to run this function again.

In the case we didn’t receive a command yet, we will set the isStuckAndWaitingForCommand to false, since we are now no longer stuck.

After that we will get back a direction without an obstacle from the “avoidObstacles()” function. I will explain this function in a bit.

We then drive to this direction for 500ms using the “drive(directionToGo, 500)” we explained before in “Motor.h”.  
  
At the end we return false to tell the timer not to run this function again.

Text, letter

Description automatically generated

The avoidObstacles() is a relatively big function so I will split it up into segments.

This function returns the direction that does not have an obstacle by using the Ultrasonic module. It is accepting a boolean as a parameter (force) which is false by default.

This parameter (force) can be used to trigger this function even if there is no obstacle in front of the robot. This is for example used in “Driver.h” with the “iAmLost()” function, so we can make the robot look left and right and go into a random direction (without an obstacle) using this function.

avoidObstacles() will return an int with the direction that does not have an obstacle.

* 0 means no direction is possible, we are stuck
* 1 means we can move forward
* 2 means we can go to the left
* 3 means we can go to the right

At the start of this function, we will use the “isStuckAndWaitingForCommand” boolean to determine if we are currently waiting for a voice command, as explained before.

When this is the case, we return and stop the code from executing further.

We create a new boolean “hasObstacle” which uses the “detectObstacle()” function explained before to determine if there is an obstacle in front of us (since we are looking forward by default).

When this is true we will stop the robot using “drive(0)” and then the robot will look if it can either go left or right. This is also the case when the “force” parameter is set to true, even if there is no obstacle in front of the robot, it will stop and look in both ways to determine any obstacles. **This is for the robot to exhibit spontaneous wandering behavior. When he is not being guided, he feels more uncertain and shows it by being more attentive looking around.**

We now create another variable, “randomInt”, which uses the function “random(2,4)” to give us back a random value from 2 to 4 (it will return either a 2, or a 3).

This randomInt is used to decide which direction we look first randomly. **This way the robot never always looks left first and then right**, it can also look right first and then left.

Graphical user interface, text, application

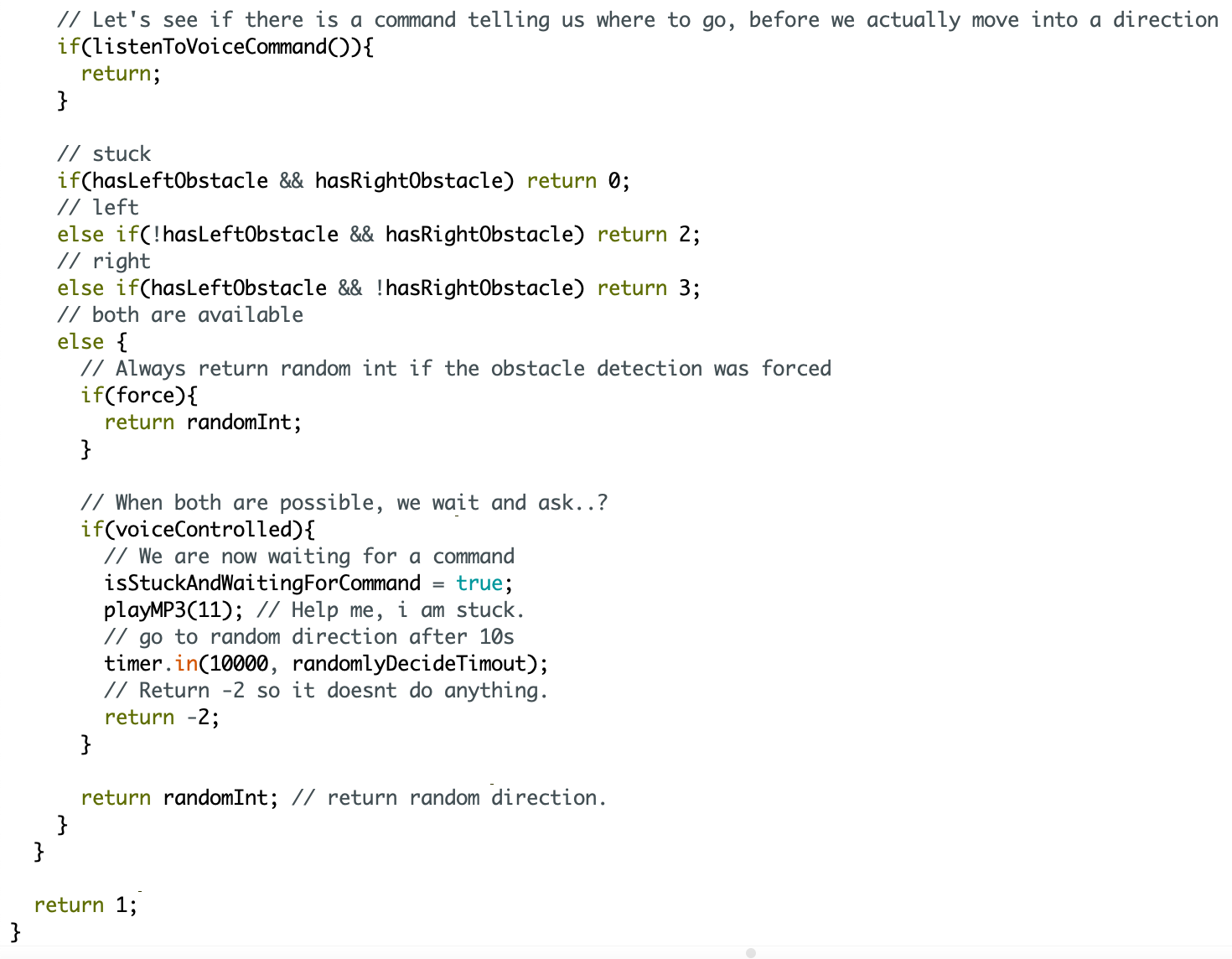
Description automatically generated

We define 2 boolean variables, hasLeftObstacle and hasRightObstacle that are false by default. These variables will be set to “true” if there is an obstacle in either of those directions.

If the randomInt we defined before returns 2, we first look left using the “lookLeft()” function. We then do an if check on the “detectObstacle()” function, which will return true or false depending if there is an obstacle in front of the robot or not. If this returns true, we will set the “hasLeftObstacle” boolean to true. We then look right using the “lookRight()” and do exactly the same, if we detect an obstacle we set the “hasRightObstacle” to true.

If the randomInt we defined before returns 3, we do exactly the same but look to the right first, and then to the left.

When this is done, the robot will still be looking either to the left or right, so we trigger the “lookForward()” function to make it look forward again. We pass “false” along as a parameter since we don’t have to wait till the Servo is done rotating the head back so it looks forward again.



Here we use the “listenToVoiceCommand()” that is defined in “VoiceCommand.h”, this allows us to listen to an incoming voice command before the robot decides on its own. When this function returns “true” a voice command was given, so we return and stop the rest of the code from executing.

Now we have to determine whether we are stuck, whether can go to the left, whether can go to the right, or whether we can go to either the left or the right.

* If both hasLeftObstacle and hasRightObstacle are true, we are **stuck**, so we return 0 (stuck).
* If hasLeftObstacle is false, and hasRightObstacle is true, we can **go to the** **left**, so we return 2 (left).
* If hasLeftObstacle is true, and hasRightObstacle is false, we can **go to the right**, so we return 3 (right).
* If none of those are the case, we can go both left or right.

If we force the Arduino to look left or right (through the “iAmLost()” function), or when the robot is not voice controlled, we return the randomInt from before. We make it so that the robot can randomly decide to go left or right, and then the return stops the function from going further.

If we are currently voice controlled, we want to **wait for a command to tell us where to go**.

To do this we set the “isStuckAndWaitingForCommand” to true, so the main loop can’t run this function (avoidObstacle()) completely anymore.

After that we will use the MP3 player to play the file that says, “Help me, I am stuck” (file 11) using “playMP3(11)”.

By starting a timer using “timer.in(10000, randomlyDecideTimeout)” we will tell the robot to decide a direction on its own if a direction was not given in 10s. (See the “randomlyDecideTimeout” function explained before.).

We then return -2, this is simply a value that does nothing, since -2 is not bound to any direction. This makes it so that the “drive()” function in “Driver.h” is not able to drive in any direction, which means we stay in the current position. Note: This is an int type of function, which implies that we only are allowed to return integer numbers. In C++ it is not the case that we can return nothing on a function that was defined to be retuning integers. For that reason, even though -2 is meaningless, it the only way to tell the function to return “nothing” and stop it there.

At the end of the function, we return 1, this is when both “force” and “hasObstacle” are false. In other words, if we are not forced to look in both directions, and we don’t have any obstacles ahead of us, then we return 1, and so the “drive()” function (within the selfDriving() function within the Driver.h file) will make the robot drive forwards.