TABLE OF CONTENTS

SL. NO.	TITLE	PAGE
1.	ABSTRACT	3
2.	INTRODUCTION	4
3.	SOFTWARE	5
4.	WORKING PRINCIPLE	6
5.	CODE	7
6.	OUTPUT	11
7.	CONCLUSION	12
8.	REFERENCES	13

ABSTRACT

Robots have numerous advantages:

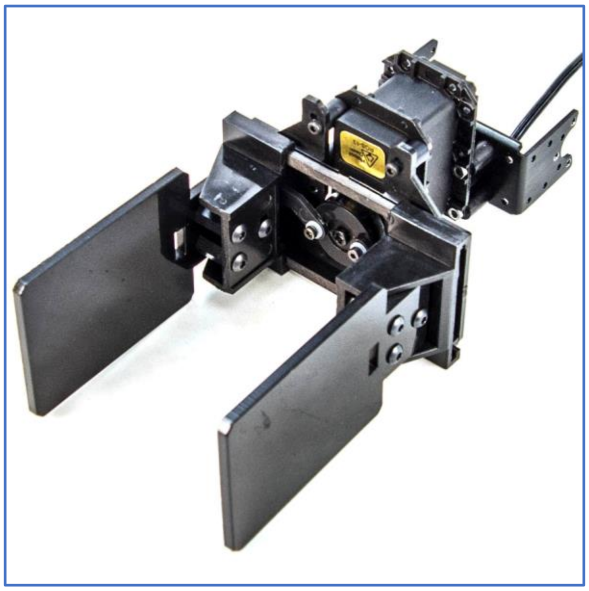
- They ensure safety, as it can be replaced in environments which are hazardous to humans.
- They are reliable because they have very less scope of error.
- They are more efficient because it can be run 24x7 without any inconvenience.
- In the long run, it turns out to be a very cost-effective solution as it does not require as much maintenance and it can perform tasks quite effortlessly.

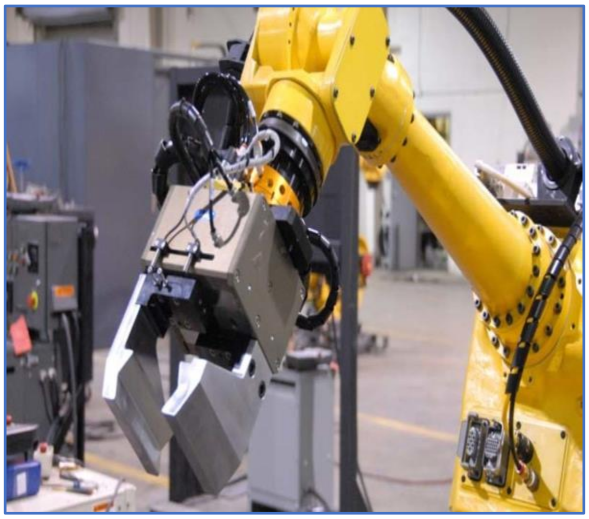
We aim to develop a *Khepera Gripper*, mounted on a twi-wheeled robot which would be mounted by multiple sensors including 9 distance sensors. This gripper would enable holding, tightening, handling and releasing of an object. It can then be attached to a robot or it can be part of a fixed automation system. The robot can move inside an arena while avoiding the walls.

The gripper would have two degrees of freedom and the arm would be moved by a DC motor coupled with a position sensor (potentiometer) which will provide absolute positioning. The regulation will be made by the microcontroller in the base board. The maximum positions (limits of movement) of the arm are limited by the software, the two limits positions (High and Ground) would be saved in the EEPROM of the microcontroller as soon as a search limits function is made.

INTRODUCTION

A gripper is a device which enables the holding of an object to be manipulated. The easiest way to describe a gripper is to think of a human hand. Grippers are used in automation systems and extensively for picking, gripping and carrying different types of objects. The gripper being developed in this project can be fixed onto a two-wheeled robot which will also be capable of obstacle avoidance.

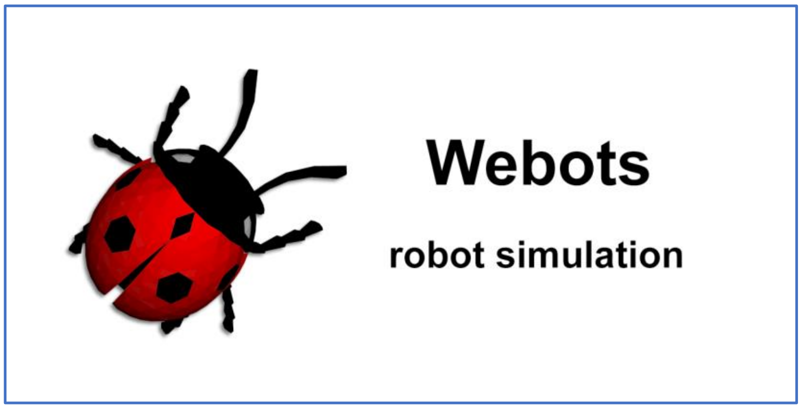




The grippers add an advantage on simple bot and make it more functional over any normal bot. Robot grippers with two fingers are the simplest robot grippers, suitable for many industrial products and easy to manufacture. Within this group, different alternatives can be found – with opening control, pressure control, with distance control in the opening and closing, picking up pieces by inserting the two fingers inside a hole. They can also have pneumatic or electric actuation.

SOFTWARE

Webots is an open source and multi-platform desktop application used to simulate robots. It provides a complete development environment to model, program and simulate robots. It has been designed for a professional use, and it is widely used in industry, education and research.



Webots includes a large collection of freely modifiable models of robots, sensors, actuators and objects. In addition, it is also possible to build new models from scratch or import them from 3D CAD software. When designing a robot model, the user specifies both the graphical and the physical properties of the objects. The graphical properties include the shape, dimensions, position and orientation, colours, and texture of the object. The physical properties include the mass, friction factor, as well as the spring and damping constants. Simple fluid dynamics is present in the software.

Webots includes a set of sensors and actuators frequently used in robotic experiments, e.g. lidars, radars, proximity sensors, light sensors, touch sensors, GPS, accelerometers, cameras, emitters and receivers, servo motors (rotational & linear), position and force sensor, LEDs, grippers, gyros, compass, IMU, etc. The robot controller programs can be written outside of Webots in C, C++, Python, ROS, Java and MATLAB using a simple API.

Webots offers the possibility to take screenshots and record simulations movies. Webots worlds are stored in cross-platform .wbt files which format is based on the VRML language. It is also possible to import and export Webots worlds or objects in the VRML format. Users can interact with a running simulation at any time, i.e., it is possible to move the robots and other object with the mouse while the simulation is running. Webots can stream a simulation on web browsers using WebGL.

WORKING PRINCIPLE

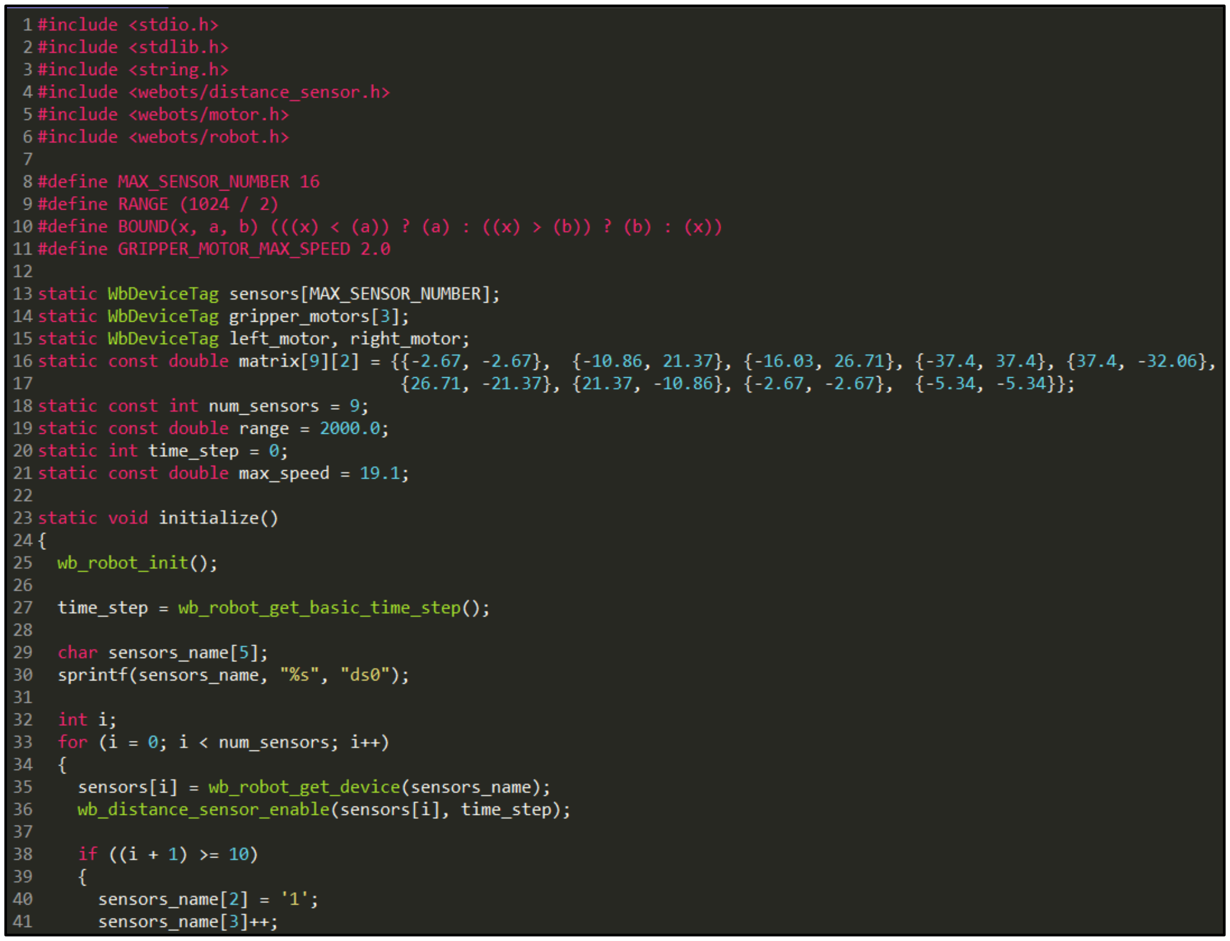
Obstacle avoidance in mobile robots refers to a robot's ability to autonomously adjust its path in response to objects in its environment. This is achieved either through proximal (reactive) or distal (rule-based) control. The difference between these two control paradigms can be subtle.

Under proximal control, the robot's sensors are tied directly to the motor controls and the motor speeds respond to the sensor input directly. One way to implement this is to create a weighted matrix that converts the sensor inputs into motor speeds. You can use a two-dimensional array with the number of rows corresponding to the number of distance sensors and the number of columns corresponding to the number of motors. For example, if we consider a two-wheel differential-wheels robot with eight sensors, then the array would have eight rows of two columns each. In this case each row can be thought of as a two item list specifying the left and right motor speeds respectively. This array is then used in the avoidance algorithm. The algorithm used as well as the values of the weighted matrix depends on the particular implementation that you are planning. The weights of the matrix are determined empirically. For example the Braitenberg algorithm below.

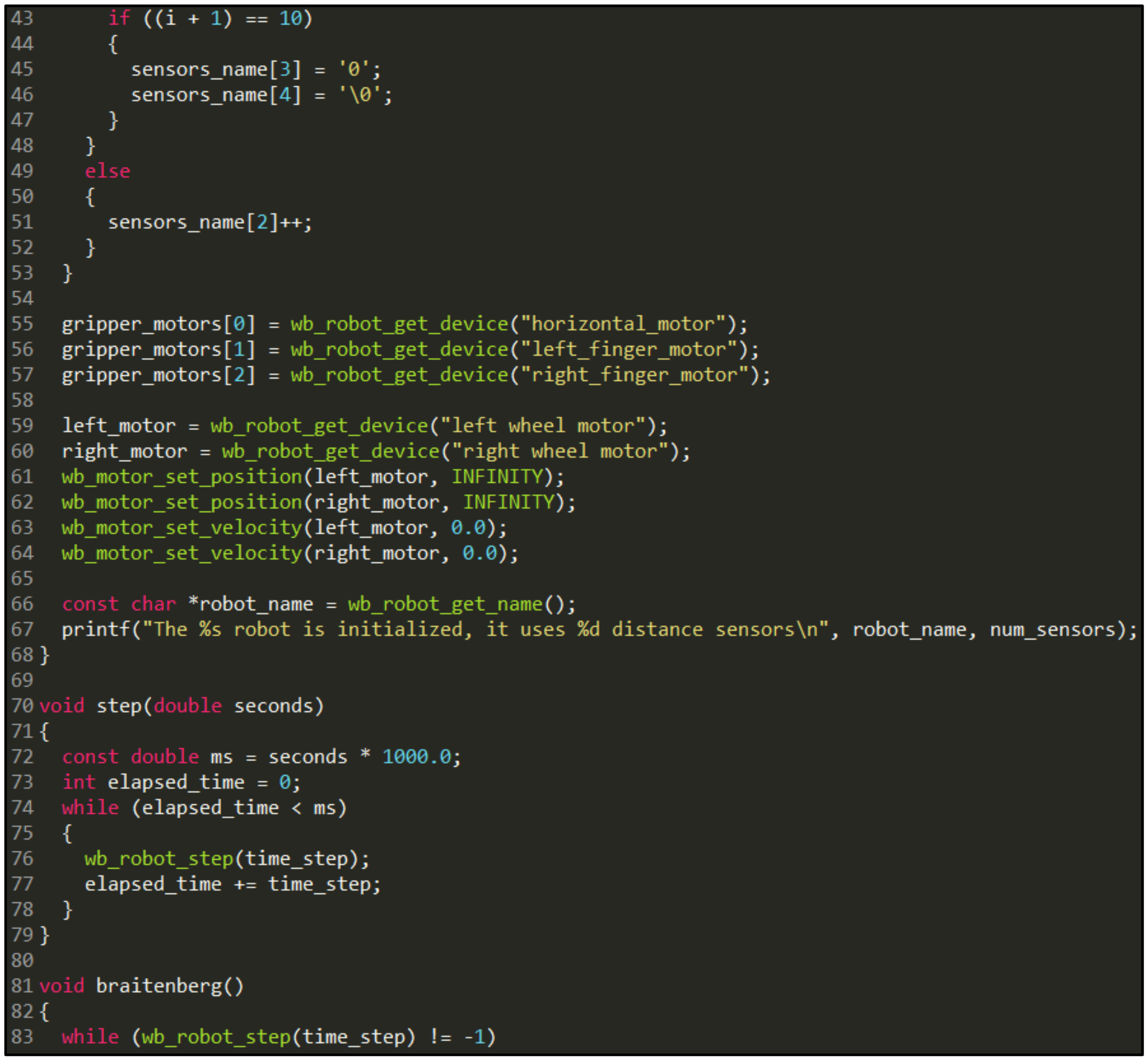
```
speed[i] += matrix[j][i] * (1.0 – (sensorsValue[j] / 512));
```

The *speed[]* array is a two item array representing the speed of the left and right motor, the *matrix[]* array holds the values in the weighted matrix, and the *sensorsValue[]* array holds the sensor values. The 512 normalizes the sensor values and the values *i* and *j* are values from a nested for loop that is used to iterate over the sensor values and weighted matrix.

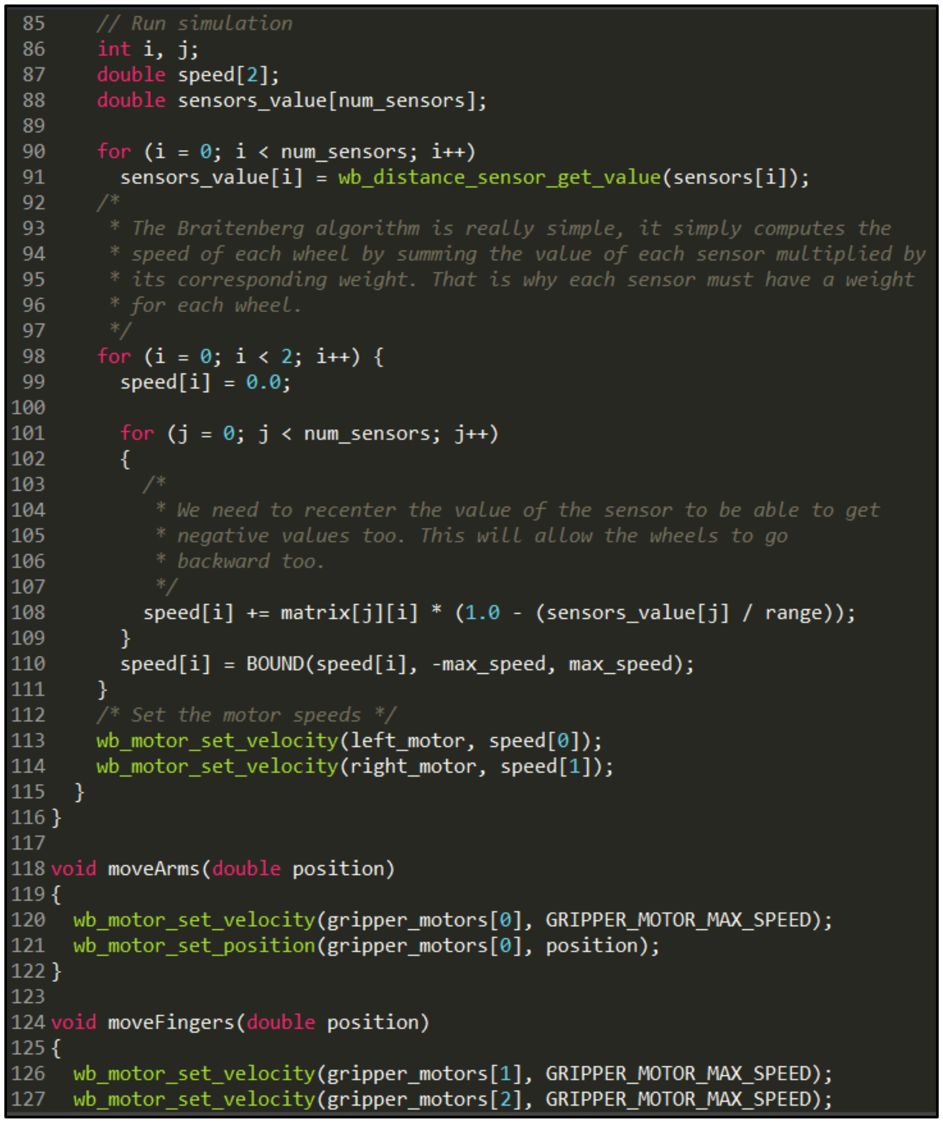
CODE



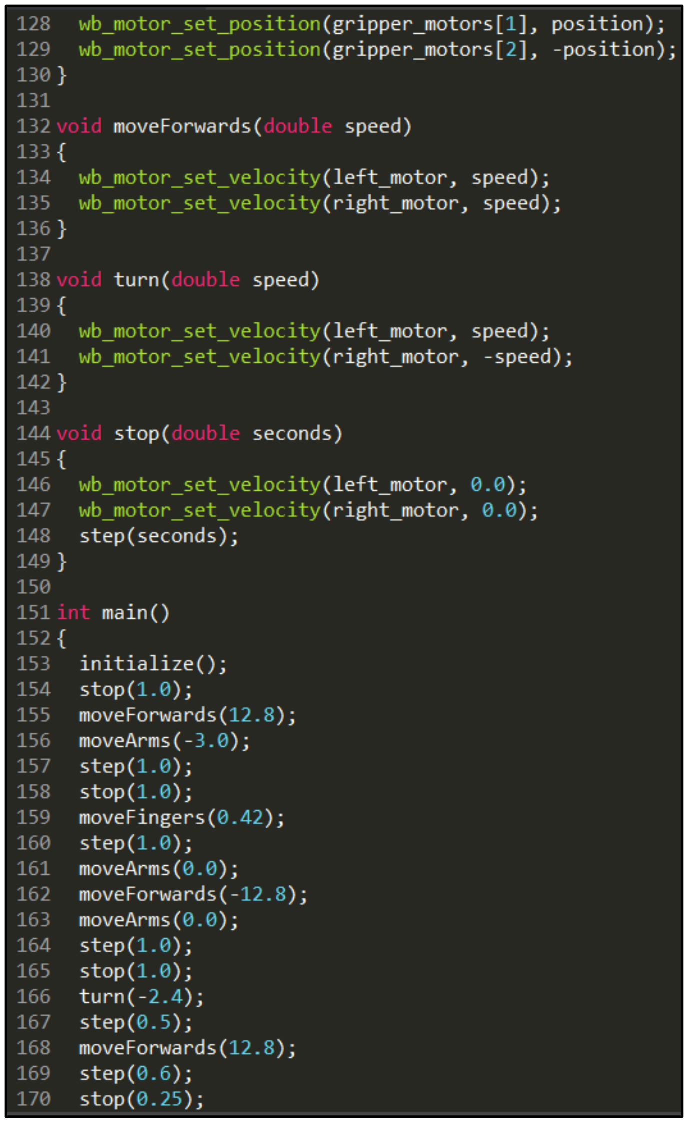
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <webots/distance_sensor.h>
#include <webots/motor.h>
#include <webots/robot.h>

#define MAX_SENSOR_NUMBER 16
#define RANGE (1024 / 2)
#define BOUND(x, a, b) (((x) < (a)) ? (a) : ((x) > (b)) ? (b) : (x))
#define GRIPPER_MOTOR_MAX_SPEED 2.0

static WbDeviceTag sensors[MAX_SENSOR_NUMBER];
static WbDeviceTag gripper_motors[3];
static WbDeviceTag left_motor, right_motor;
static const double matrix[9][2] = {{-2.67, -2.67},  {-10.86, 21.37}, {-16.03, 26.71}, {-37.4, 37.4}, {37.4, -32.06},
                                    {26.71, -21.37}, {21.37, -10.86}, {-2.67, -2.67},  {-5.34, -5.34}};
static const int num_sensors = 9;
static const double range = 2000.0;
static int time_step = 0;
static const double max_speed = 19.1;

static void initialize()
{
  wb_robot_init();

  time_step = wb_robot_get_basic_time_step();

  char sensors_name[5];
  sprintf(sensors_name, "%s", "ds0");

  int i;
  for (i = 0; i < num_sensors; i++)
  {
    sensors[i] = wb_robot_get_device(sensors_name);
    wb_distance_sensor_enable(sensors[i], time_step);

    if ((i + 1) >= 10)
    {
      sensors_name[2] = '1';
      sensors_name[3]++;
```



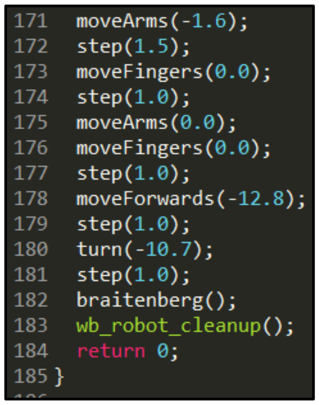
```c
      if ((i + 1) == 10)
      {
        sensors_name[3] = '0';
        sensors_name[4] = '\0';
      }
    }
    else
    {
      sensors_name[2]++;
    }
  }

  gripper_motors[0] = wb_robot_get_device("horizontal_motor");
  gripper_motors[1] = wb_robot_get_device("left_finger_motor");
  gripper_motors[2] = wb_robot_get_device("right_finger_motor");

  left_motor = wb_robot_get_device("left wheel motor");
  right_motor = wb_robot_get_device("right wheel motor");
  wb_motor_set_position(left_motor, INFINITY);
  wb_motor_set_position(right_motor, INFINITY);
  wb_motor_set_velocity(left_motor, 0.0);
  wb_motor_set_velocity(right_motor, 0.0);

  const char *robot_name = wb_robot_get_name();
  printf("The %s robot is initialized, it uses %d distance sensors\n", robot_name, num_sensors);
}

void step(double seconds)
{
  const double ms = seconds * 1000.0;
  int elapsed_time = 0;
  while (elapsed_time < ms)
  {
    wb_robot_step(time_step);
    elapsed_time += time_step;
  }
}

void braitenberg()
{
  while (wb_robot_step(time_step) != -1)
```



```c
    // Run simulation
    int i, j;
    double speed[2];
    double sensors_value[num_sensors];

    for (i = 0; i < num_sensors; i++)
      sensors_value[i] = wb_distance_sensor_get_value(sensors[i]);
    /*
     * The Braitenberg algorithm is really simple, it simply computes the
     * speed of each wheel by summing the value of each sensor multiplied by
     * its corresponding weight. That is why each sensor must have a weight
     * for each wheel.
     */
    for (i = 0; i < 2; i++) {
      speed[i] = 0.0;

      for (j = 0; j < num_sensors; j++)
      {
        /*
         * We need to recenter the value of the sensor to be able to get
         * negative values too. This will allow the wheels to go
         * backward too.
         */
        speed[i] += matrix[j][i] * (1.0 - (sensors_value[j] / range));
      }
      speed[i] = BOUND(speed[i], -max_speed, max_speed);
    }
    /* Set the motor speeds */
    wb_motor_set_velocity(left_motor, speed[0]);
    wb_motor_set_velocity(right_motor, speed[1]);
  }
}

void moveArms(double position)
{
  wb_motor_set_velocity(gripper_motors[0], GRIPPER_MOTOR_MAX_SPEED);
  wb_motor_set_position(gripper_motors[0], position);
}

void moveFingers(double position)
{
  wb_motor_set_velocity(gripper_motors[1], GRIPPER_MOTOR_MAX_SPEED);
  wb_motor_set_velocity(gripper_motors[2], GRIPPER_MOTOR_MAX_SPEED);
```



```c
  wb_motor_set_position(gripper_motors[1], position);
  wb_motor_set_position(gripper_motors[2], -position);
}

void moveForwards(double speed)
{
  wb_motor_set_velocity(left_motor, speed);
  wb_motor_set_velocity(right_motor, speed);
}

void turn(double speed)
{
  wb_motor_set_velocity(left_motor, speed);
  wb_motor_set_velocity(right_motor, -speed);
}

void stop(double seconds)
{
  wb_motor_set_velocity(left_motor, 0.0);
  wb_motor_set_velocity(right_motor, 0.0);
  step(seconds);
}

int main()
{
  initialize();
  stop(1.0);
  moveForwards(12.8);
  moveArms(-3.0);
  step(1.0);
  stop(1.0);
  moveFingers(0.42);
  step(1.0);
  moveArms(0.0);
  moveForwards(-12.8);
  moveArms(0.0);
  step(1.0);
  stop(1.0);
  turn(-2.4);
  step(0.5);
  moveForwards(12.8);
  step(0.6);
  stop(0.25);
```

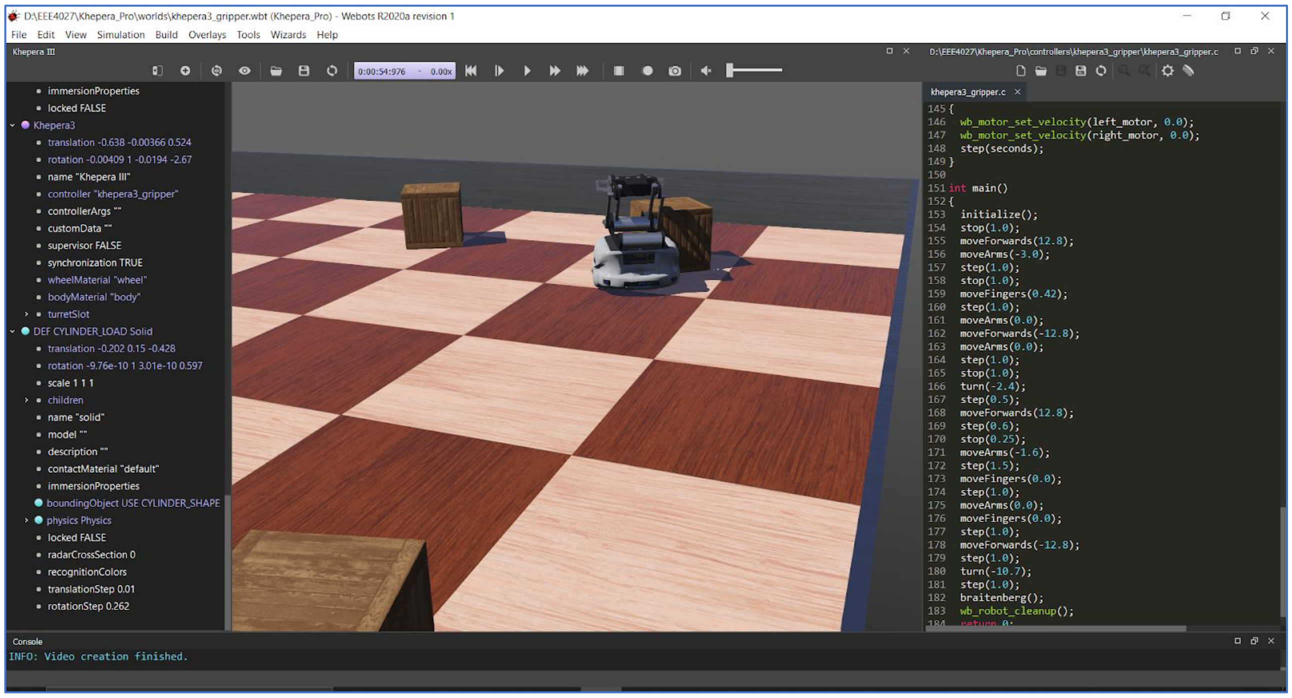


```c
  moveArms(-1.6);
  step(1.5);
  moveFingers(0.0);
  step(1.0);
  moveArms(0.0);
  moveFingers(0.0);
  step(1.0);
  moveForwards(-12.8);
  step(1.0);
  turn(-10.7);
  step(1.0);
  braitenberg();
  wb_robot_cleanup();
  return 0;
}
```

OUTPUT

Here is the YouTube video link for the Webots simulation –

Simulation Of Khepera Gripper Bot


D:\EEE4027\Khepera_Pro\worlds\khepera3_gripper.wbt (Khepera_Pro) - Webots R2020a revision 1
File Edit View Simulation Build Overlays Tools Wizards Help
Khepera III
0:00:54:976
0.00x
D:\EEE4027\Khepera_Pro\controllers\khepera3_gripper\khepera3_gripper.c
khepera3_gripper.c
Console

```
    • immersionProperties
    • locked FALSE
⌄ ● Khepera3
    • translation -0.638 -0.00366 0.524
    • rotation -0.00409 1 -0.0194 -2.67
    • name "Khepera III"
    • controller "khepera3_gripper"
    • controllerArgs ""
    • customData ""
    • supervisor FALSE
    • synchronization TRUE
    • wheelMaterial "wheel"
    • bodyMaterial "body"
  › • turretSlot
⌄ ● DEF CYLINDER_LOAD Solid
    • translation -0.202 0.15 -0.428
    • rotation -9.76e-10 1 3.01e-10 0.597
    • scale 1 1 1
  › • children
    • name "solid"
    • model ""
    • description ""
    • contactMaterial "default"
    • immersionProperties
    ● boundingObject USE CYLINDER_SHAPE
  › ● physics Physics
    • locked FALSE
    • radarCrossSection 0
    • recognitionColors
    • translationStep 0.01
    • rotationStep 0.262
```

```c
{
  wb_motor_set_velocity(left_motor, 0.0);
  wb_motor_set_velocity(right_motor, 0.0);
  step(seconds);
}

int main()
{
  initialize();
  stop(1.0);
  moveForwards(12.8);
  moveArms(-3.0);
  step(1.0);
  stop(1.0);
  moveFingers(0.42);
  step(1.0);
  moveArms(0.0);
  moveForwards(-12.8);
  moveArms(0.0);
  step(1.0);
  stop(1.0);
  turn(-2.4);
  step(0.5);
  moveForwards(12.8);
  step(0.6);
  stop(0.25);
  moveArms(-1.6);
  step(1.5);
  moveFingers(0.0);
  step(1.0);
  moveArms(0.0);
  moveFingers(0.0);
  step(1.0);
  moveForwards(-12.8);
  step(1.0);
  turn(-10.7);
  step(1.0);
  braitenberg();
  wb_robot_cleanup();
  return 0;
```

```
INFO: Video creation finished.
```

(To help the gripper differentiate between the object that needs to be picked up and the wooden boxes in the environment, the mass of the wooden boxes has been set to zero.)

CONCLUSION

Robot grippers are the physical interface between a robot arm and the work piece. This end-of-arm tooling (EOAT) is one of the most important parts of the robot. One of the many benefits of material handling robots is the reduction of part damage. A gripper comes in direct contact with your product, so it's important to choose the right type of gripper for your operation.

To conclude the robots can carry out the following:

4Ds

- Can carry out **dangerous** tasks
- Can carry out **dirty** tasks
- Can carry out **dull** tasks
- Can carry out **difficult** tasks

4As

- Can be used to **automate**
- Can be used to **augment** humans
- Can be used to **assist** humans
- Can be used as **autonomous** entity

REFERENCES

1. Yang, Xiaoyu, Rajnikant V. Patel, and Mehrdad Moallem. "A fuzzy–braitenberg navigation strategy for differential drive mobile robots." *Journal of Intelligent and Robotic Systems* 47.2 (2006): 101-124.

2. Michel, Olivier. "Cyberbotics Ltd. Webots™: professional mobile robot simulation." *International Journal of Advanced Robotic Systems* 1.1 (2004): 5.

3. Michel, Olivier. "Webots: Symbiosis between virtual and real mobile robots." *International Conference on Virtual Worlds*. Springer, Berlin, Heidelberg, 1998.

List of web references –

- Robotics Lab Mannual - University of Colorado
- WeBots User Guide
- RobotWorx – Grippers For Robots