Приложение 1. Модель платформы робота культиватора разработанной в ПО «AutoCAD»

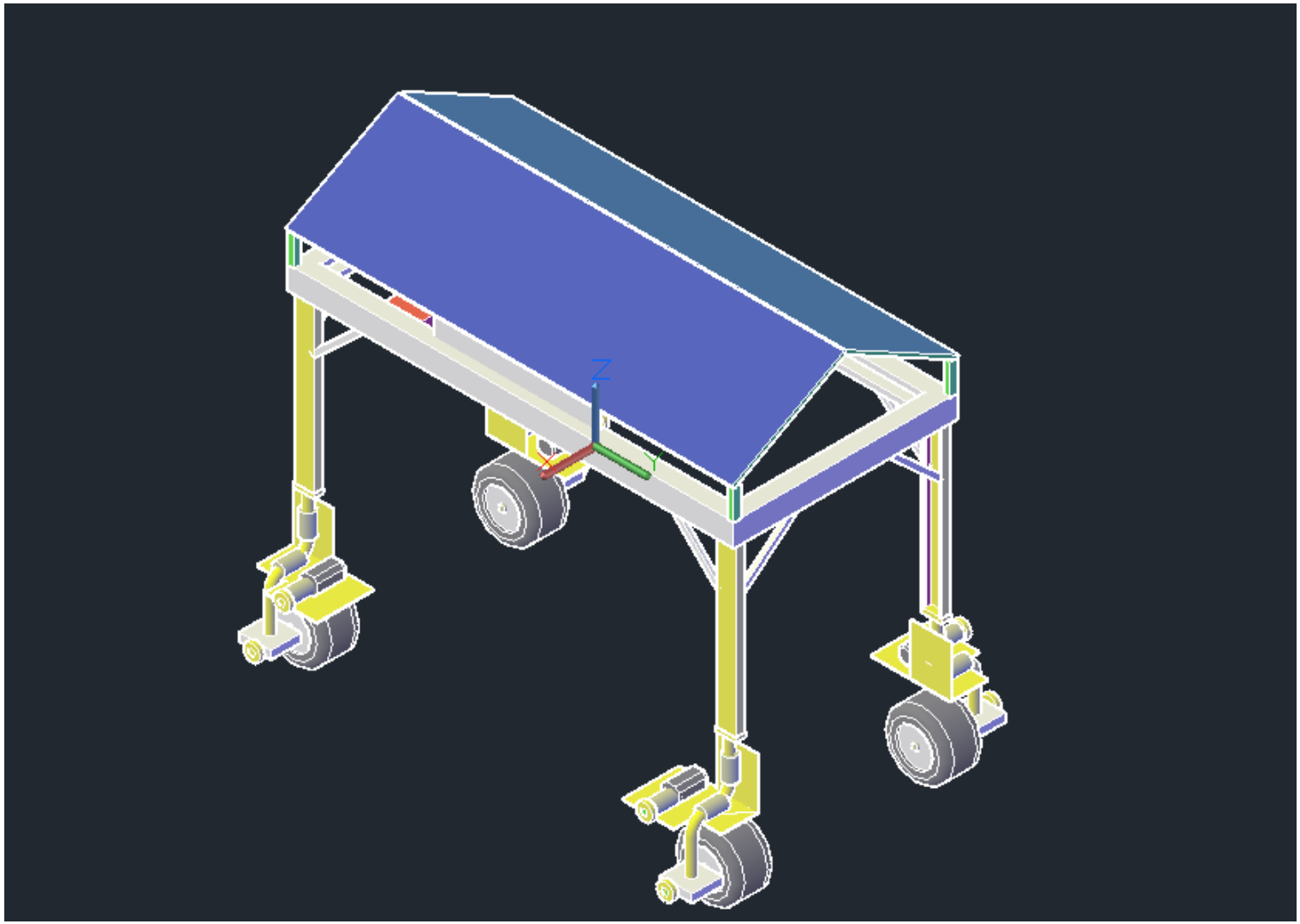

Рисунок П.1.1. – Модель платформы робота культиватора

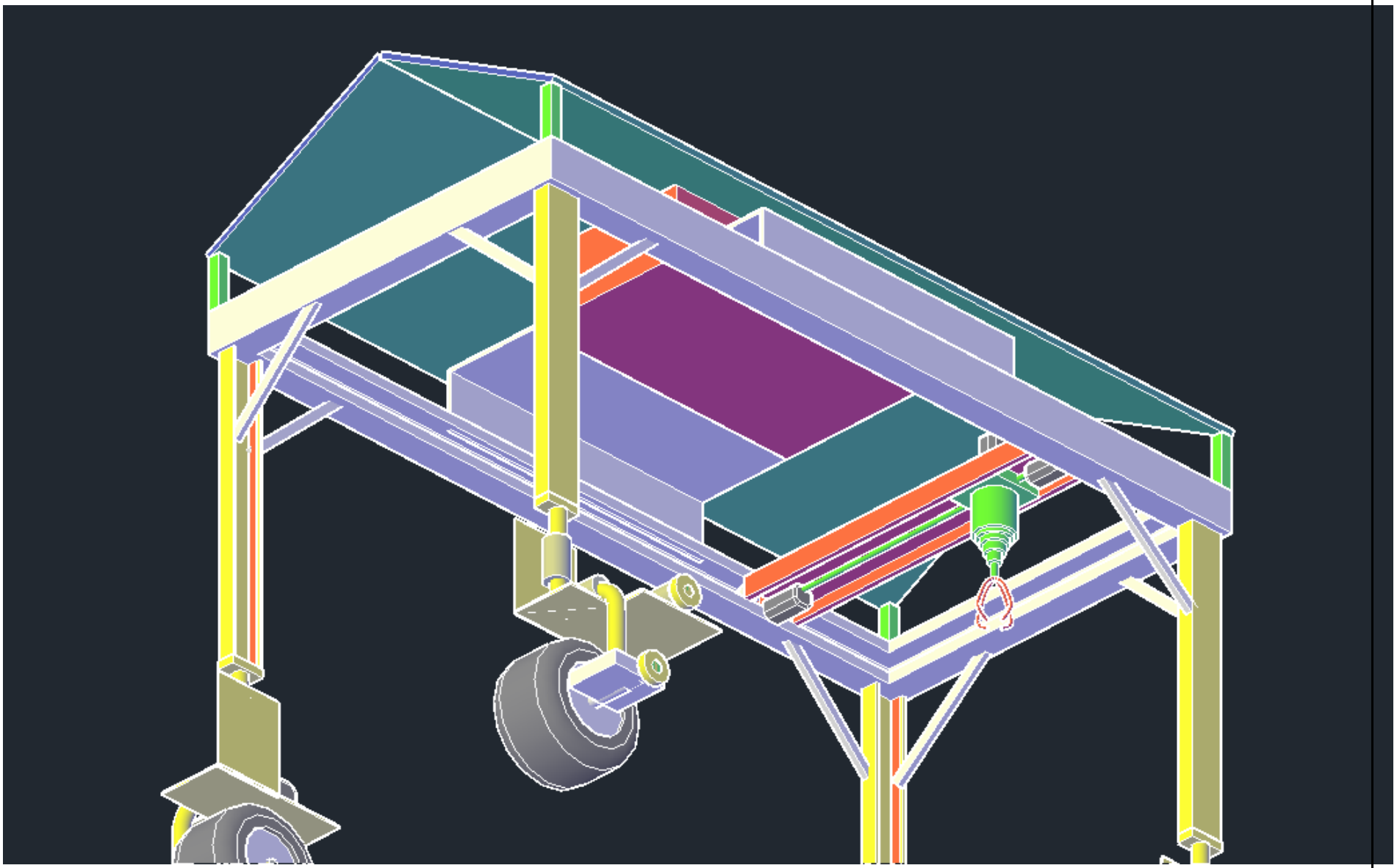

Рисунок П.1.2. – Модель платформы робота культиватора

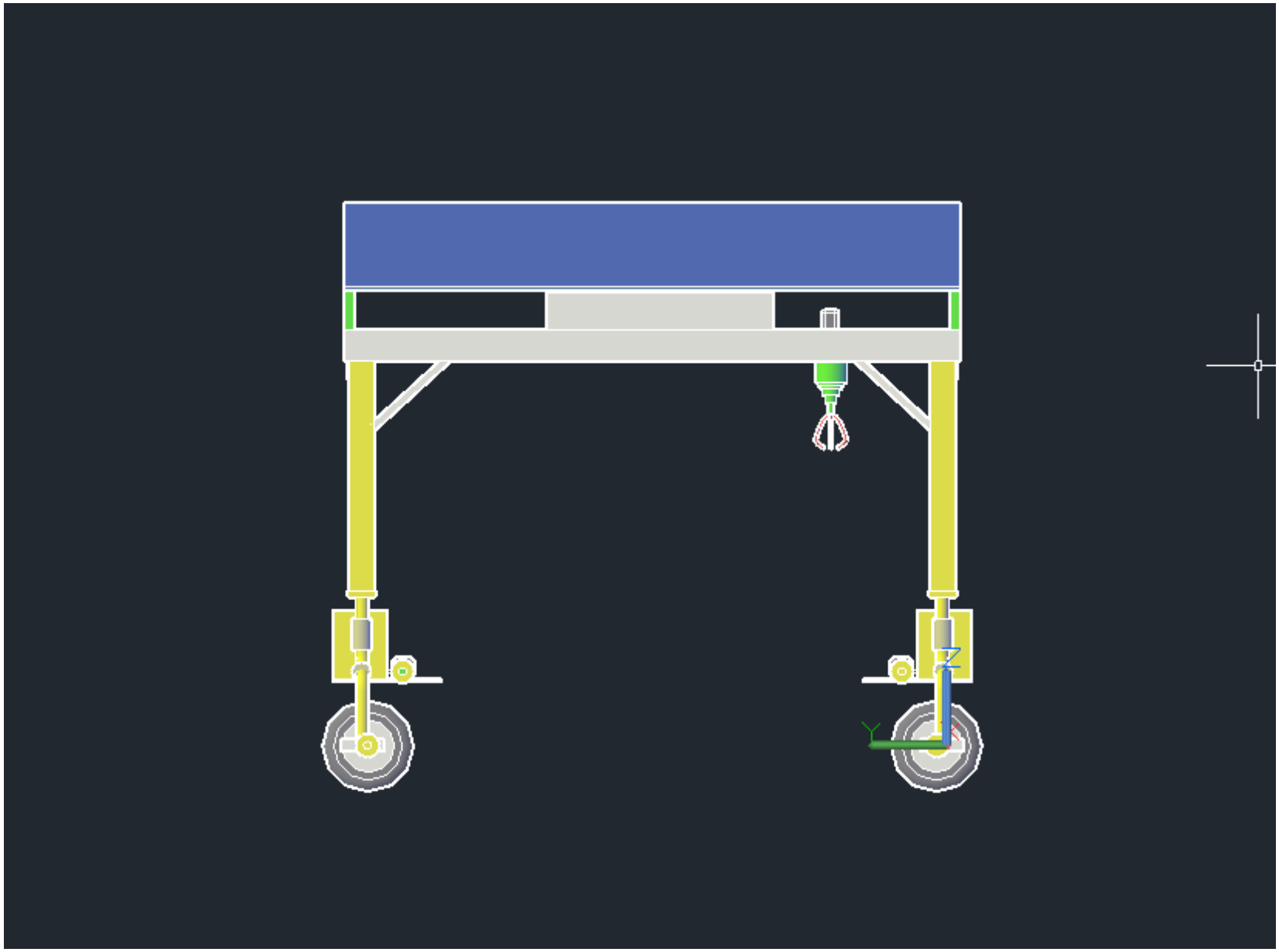

Рисунок П.1.3. – Модель платформы робота культиватора

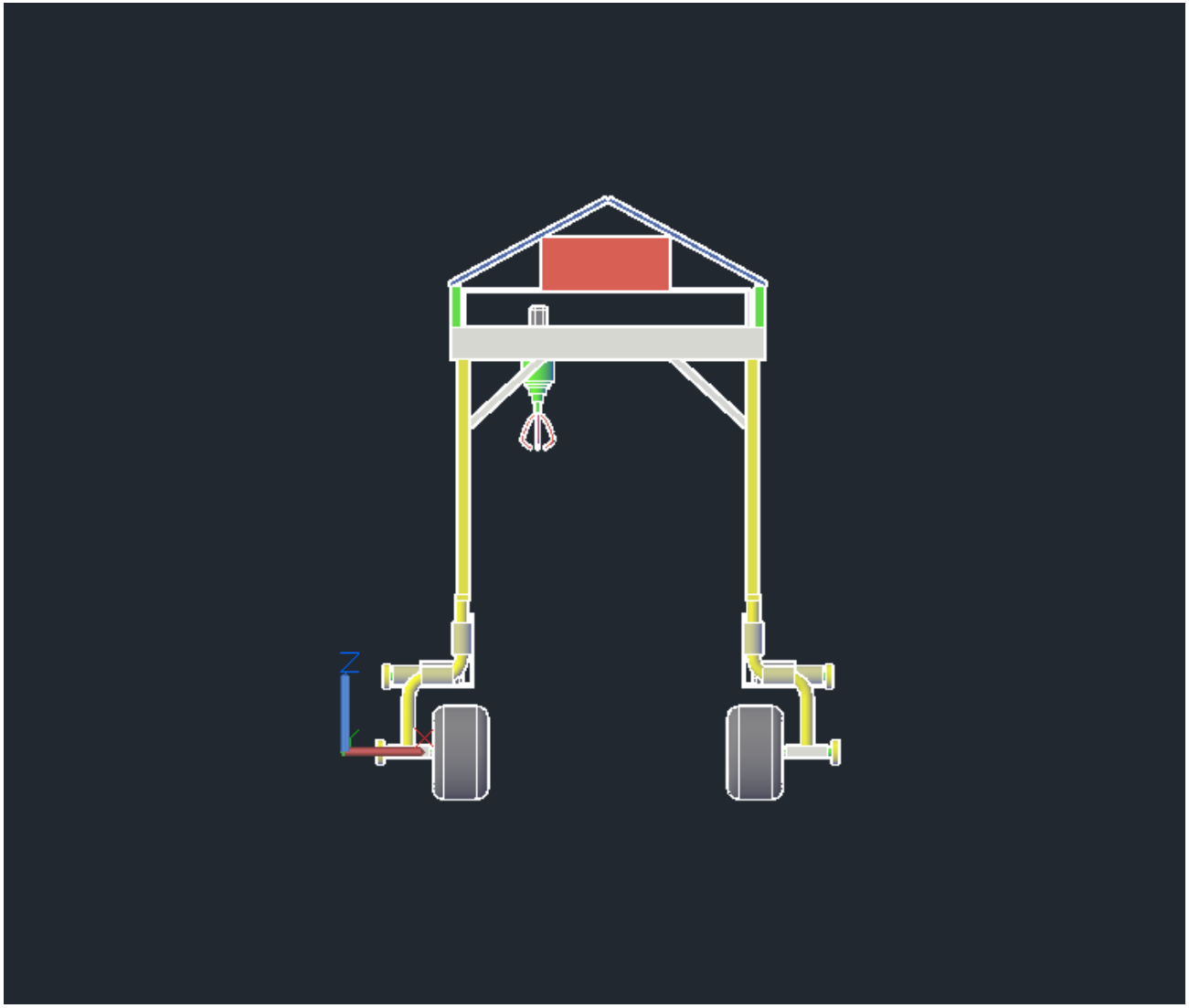

Рисунок П.1.4. – Модель платформы робота культиватора

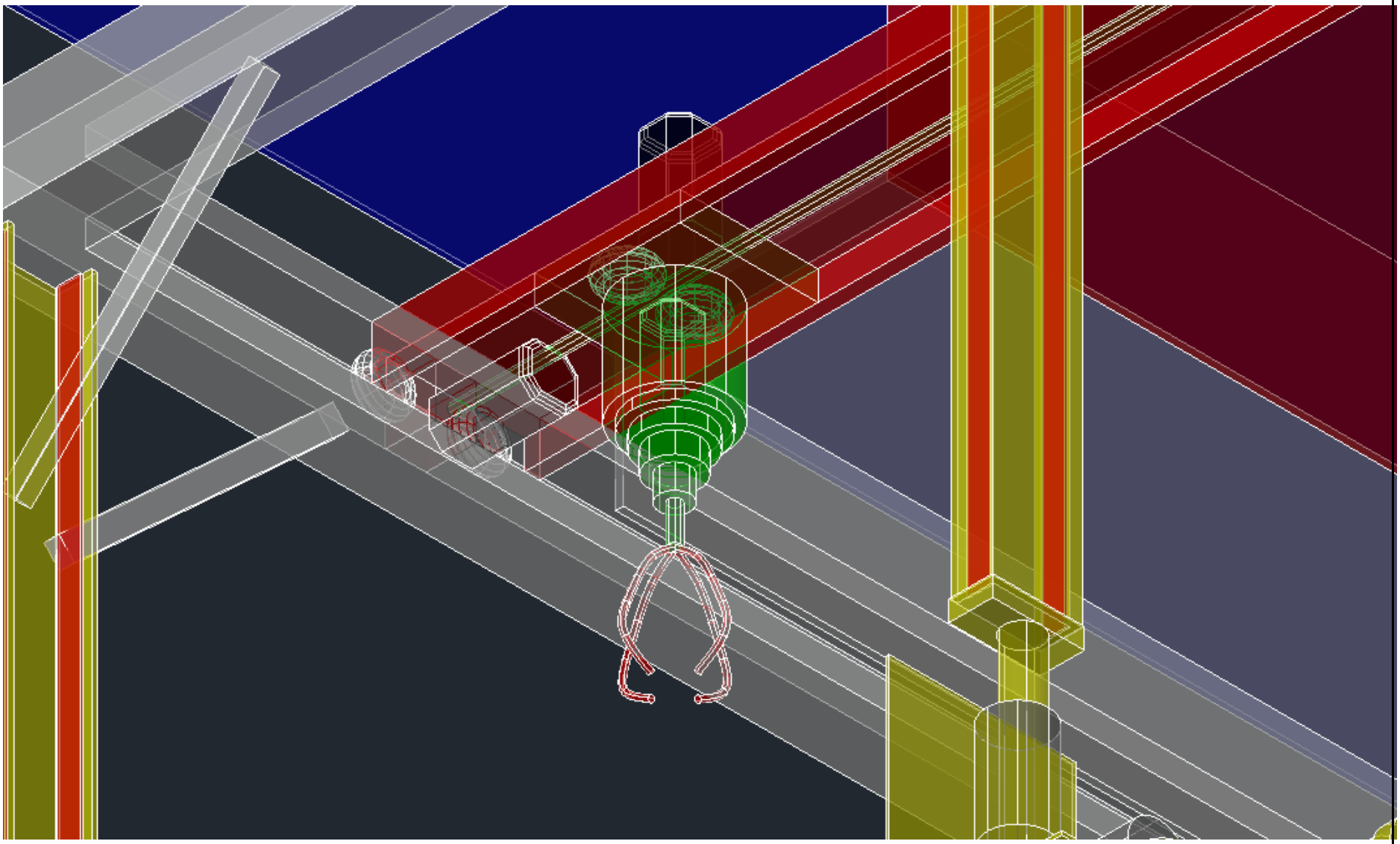

Рисунок П.1.5. – Модель платформы робота культиватора

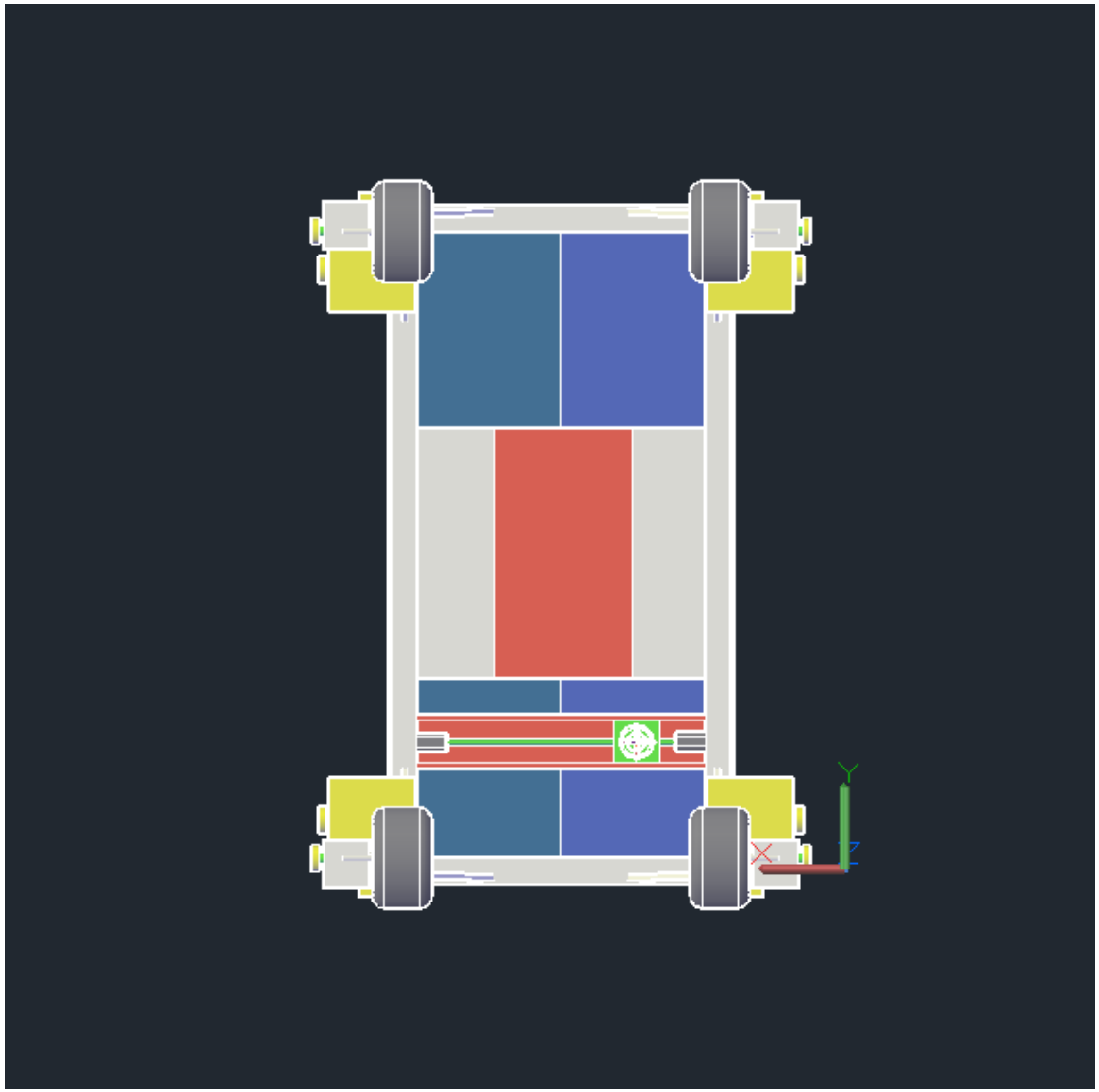

Рисунок П.1.6. – Модель платформы робота культиватора

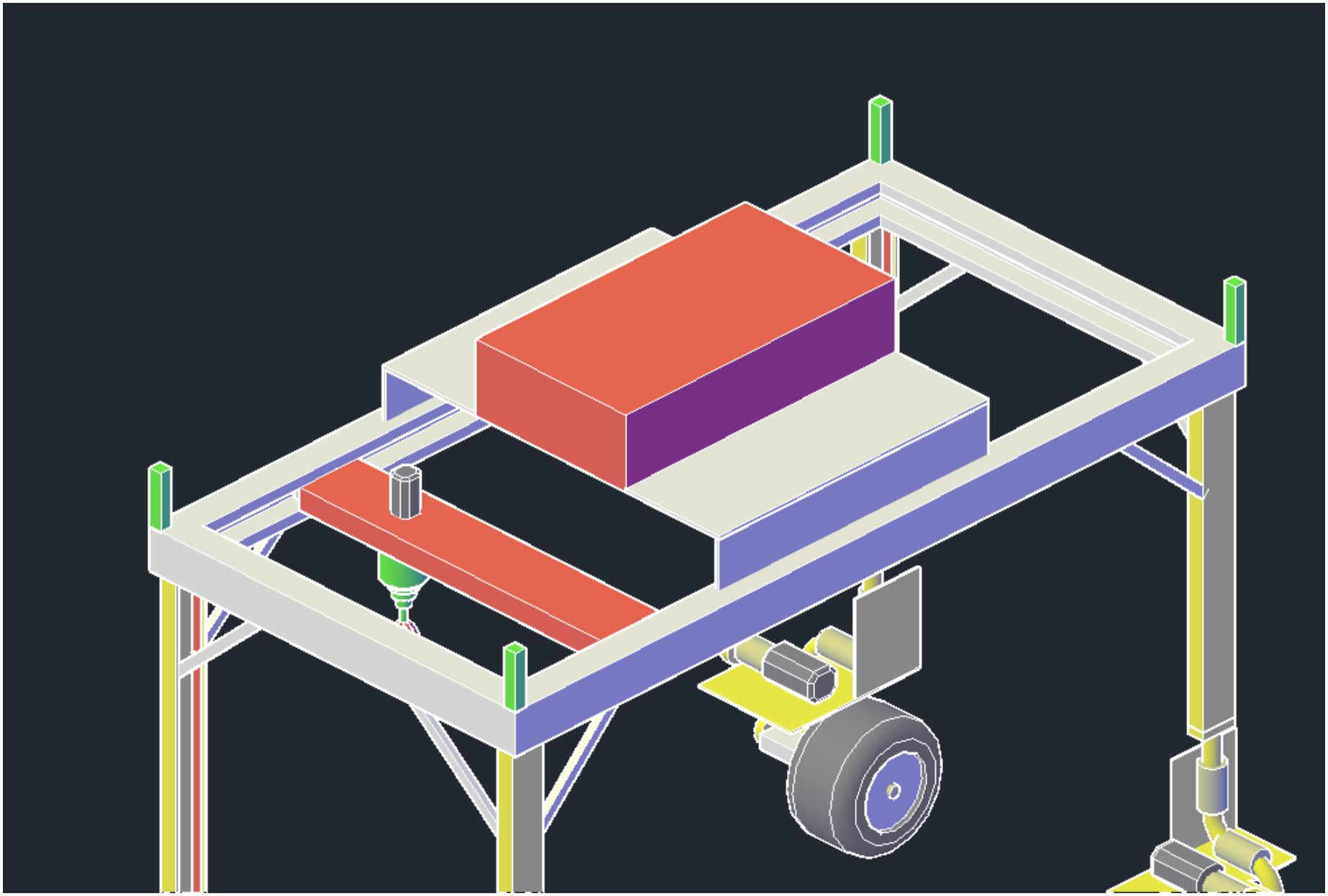

Рисунок П.1.7. – Модель платформы робота культиватора

Приложение 2. Листинг программы заложенный в одноплатный компьютер «Nvidia Jetson Nano» для определения сорных растении

```
######## Webcam Object Detection Using Tensorflow-trained Classifier #########
#
# Author: Evan Juras
# Date: 10/27/19
# Description:
# This program uses a TensorFlow Lite model to perform object detection on a live webcam
# feed. It draws boxes and scores around the objects of interest in each frame from the
# webcam. To improve FPS, the webcam object runs in a separate thread from the main program.
# This script will work with either a Picamera or regular USB webcam.
#
# This code is based off the TensorFlow Lite image classification example at:
#
https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/examples/python/label_image.py
#
# I added my own method of drawing boxes and labels using OpenCV.

# Import packages
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util

# Define VideoStream class to handle streaming of video from webcam in separate processing thread
# Source - Adrian Rosebrock, PyImageSearch: https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/
class VideoStream:
    """Camera object that controls video streaming from the Picamera"""
    def __init__(self,resolution=(640,480),framerate=30):
```

```
            # Initialize the PiCamera and the camera image stream
            self.stream = cv2.VideoCapture(0)
            ret = self.stream.set(cv2.CAP_PROP_FOURCC,
cv2.VideoWriter_fourcc(*'MJPG'))
            ret = self.stream.set(3,resolution[0])
            ret = self.stream.set(4,resolution[1])

            # Read first frame from the stream
            (self.grabbed, self.frame) = self.stream.read()

              # Variable to control when the camera is stopped
            self.stopped = False

        def start(self):
              # Start the thread that reads frames from the video stream
            Thread(target=self.update,args=()).start()
            return self

        def update(self):
            # Keep looping indefinitely until the thread is stopped
            while True:
                # If the camera is stopped, stop the thread
                if self.stopped:
                    # Close camera resources
                    self.stream.release()
                    return

                # Otherwise, grab the next frame from the stream
                (self.grabbed, self.frame) = self.stream.read()

        def read(self):
              # Return the most recent frame
            return self.frame

        def stop(self):
              # Indicate that the camera and thread should be stopped
            self.stopped = True

    # Define and parse input arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                        required=True)
```

```
        parser.add_argument('--graph', help='Name of the .tflite file, if different than
detect.tflite',
                            default='detect.tflite')
        parser.add_argument('--labels', help='Name of the labelmap file, if different
than labelmap.txt',
                            default='labelmap.txt')
        parser.add_argument('--threshold', help='Minimum confidence threshold for
displaying detected objects',
                            default=0.5)
        parser.add_argument('--resolution', help='Desired webcam resolution in WxH.
If the webcam does not support the resolution entered, errors may occur.',
                            default='1280x720')
        parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to
speed up detection',
                            action='store_true')

        args = parser.parse_args()

        MODEL_NAME = args.modeldir
        GRAPH_NAME = args.graph
        LABELMAP_NAME = args.labels
        min_conf_threshold = float(args.threshold)
        resW, resH = args.resolution.split('x')
        imW, imH = int(resW), int(resH)
        use_TPU = args.edgetpu

        # Import TensorFlow libraries
        # If tflite_runtime is installed, import interpreter from tflite_runtime, else
import from regular tensorflow
        # If using Coral Edge TPU, import the load_delegate library
        pkg = importlib.util.find_spec('tflite_runtime')
        if pkg:
            from tflite_runtime.interpreter import Interpreter
            if use_TPU:
                from tflite_runtime.interpreter import load_delegate
        else:
            from tensorflow.lite.python.interpreter import Interpreter
            if use_TPU:
                from tensorflow.lite.python.interpreter import load_delegate

        # If using Edge TPU, assign filename for Edge TPU model
        if use_TPU:
```

```
            # If user has specified the name of the .tflite file, use that name, otherwise
use default 'edgetpu.tflite'
            if (GRAPH_NAME == 'detect.tflite'):
                GRAPH_NAME = 'edgetpu.tflite'

        # Get path to current working directory
        CWD_PATH = os.getcwd()

        # Path to .tflite file, which contains the model that is used for object detection
        PATH_TO_CKPT =
os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

        # Path to label map file
        PATH_TO_LABELS =
os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

        # Load the label map
        with open(PATH_TO_LABELS, 'r') as f:
            labels = [line.strip() for line in f.readlines()]

        # Have to do a weird fix for label map if using the COCO "starter model" from
        # https://www.tensorflow.org/lite/models/object_detection/overview
        # First label is '???', which has to be removed.
        if labels[0] == '???':
            del(labels[0])

        # Load the Tensorflow Lite model.
        # If using Edge TPU, use special load_delegate argument
        if use_TPU:
            interpreter = Interpreter(model_path=PATH_TO_CKPT,
                                      experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
            print(PATH_TO_CKPT)
        else:
            interpreter = Interpreter(model_path=PATH_TO_CKPT)

        interpreter.allocate_tensors()

        # Get model details
        input_details = interpreter.get_input_details()
        output_details = interpreter.get_output_details()
        height = input_details[0]['shape'][1]
        width = input_details[0]['shape'][2]
```

```
        floating_model = (input_details[0]['dtype'] == np.float32)

        input_mean = 127.5
        input_std = 127.5

        # Initialize frame rate calculation
        frame_rate_calc = 1
        freq = cv2.getTickFrequency()

        # Initialize video stream
        videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
        time.sleep(1)

        #for frame1 in camera.capture_continuous(rawCapture,
format="bgr",use_video_port=True):
        while True:

            # Start timer (for calculating frame rate)
            t1 = cv2.getTickCount()

            # Grab frame from video stream
            frame1 = videostream.read()

            # Acquire frame and resize to expected shape [1xHxWx3]
            frame = frame1.copy()
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame_resized = cv2.resize(frame_rgb, (width, height))
            input_data = np.expand_dims(frame_resized, axis=0)

            # Normalize pixel values if using a floating model (i.e. if model is non-
quantized)
            if floating_model:
               input_data = (np.float32(input_data) - input_mean) / input_std

            # Perform the actual detection by running the model with the image as input
            interpreter.set_tensor(input_details[0]['index'],input_data)
            interpreter.invoke()

            # Retrieve detection results
            boxes = interpreter.get_tensor(output_details[0]['index'])[0] # Bounding box
coordinates of detected objects
```

```
        classes = interpreter.get_tensor(output_details[1]['index'])[0] # Class index
of detected objects
        scores = interpreter.get_tensor(output_details[2]['index'])[0] # Confidence of
detected objects
        #num = interpreter.get_tensor(output_details[3]['index'])[0]  # Total number
of detected objects (inaccurate and not needed)

        # Loop over all detections and draw detection box if confidence is above
minimum threshold
        for i in range(len(scores)):
           if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

               # Get bounding box coordinates and draw box
               # Interpreter can return coordinates that are outside of image
dimensions, need to force them to be within image using max() and min()
               ymin = int(max(1,(boxes[i][0] * imH)))
               xmin = int(max(1,(boxes[i][1] * imW)))
               ymax = int(min(imH,(boxes[i][2] * imH)))
               xmax = int(min(imW,(boxes[i][3] * imW)))

               cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

               # Draw label
               object_name = labels[int(classes[i])] # Look up object name from
"labels" array using class index
               label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example:
'person: 72%'
               labelSize, baseLine = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
               label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw
label too close to top of window
               cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10),
(xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw
white box to put label text in
               cv2.putText(frame, label, (xmin, label_ymin-7),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text

        # Draw framerate in corner of frame
        cv2.putText(frame,'FPS:
{0:.2f}'.format(frame_rate_calc),(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(255,2
55,0),2,cv2.LINE_AA)
```

```
        # All the results have been drawn on the frame, so it's time to display it.
        cv2.imshow('Object detector', frame)

        # Calculate framerate
        t2 = cv2.getTickCount()
        time1 = (t2-t1)/freq
        frame_rate_calc= 1/time1

        # Press 'q' to quit
        if cv2.waitKey(1) == ord('q'):
            break

    # Clean up
    cv2.destroyAllWindows()
    videostream.stop()
```

Листинг программы П.2.1. – Листинг программы для алгоритма распознавания сорных растении

Приложение 3. Исходные коды, заложенные в одноплатный компьютер «Raspberry Pi4»

```
import RPi.GPIO as GPIO
import time

# Ultra sonic distance sensor`s PIN constants
TRIGGER = 18
ECHO = 24

# Motor-1`s PIN constants
ENA = 32
IN1 = 29
IN2 = 31

# Motor-2`s PIN constants
ENB = 33
IN3 = 11
IN4 = 13

# Setting up PIN modes
GPIO.setmode(GPIO.BOARD)  # by physical PIN layout
GPIO.setwarnings(False)  # disable warnings

GPIO.setup(TRIGGER, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

GPIO.setup(ENA, GPIO.OUT)
GPIO.setup(ENB, GPIO.OUT)
GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)


ena_pwm = GPIO.PWM(ENA, 1000)  # create PWM instance with frequency
enb_pwm = GPIO.PWM(ENB, 1000)
ena_pwm.start(0)  # start PWM of required Duty Cycle
enb_pwm.start(0)


def custom_normalize(to_convert, input_range, output_range):
```

```
    """
        Parameters:
            to_convert --> number to convert from input range to output range
            input_range --> list of 2 integer of float items
            output_range --> list of 2 integer of float items
    """
    assert len(input_range) == 2, "Input range should be list of 2 items"
    assert len(output_range) == 2, "Output range should be list of 2 items"
    assert type(to_convert) == int or type(to_convert) == float, "Only integer and float
is allowed to convert"

    bipolar_ouput_range = False

    for i in input_range + output_range:
        assert type(i) == int or type(i) == float, "Only integers and floats are allowed in
range values"
        if i < 0:
            bipolar_ouput_range = True

    if bipolar_ouput_range:
        coefficient = abs((input_range[0] - input_range[1])/(output_range[0] -
output_range[1]))

        if to_convert < 0:
            return max(round(to_convert/coefficient-output_range[1], 2), output_range[0])
        return min(round(to_convert/coefficient-output_range[1], 2), output_range[1])
    else:
        coefficient = abs((input_range[0] - input_range[1])/(output_range[0] -
output_range[1]))

        if to_convert < 0:
            return max(round(to_convert/coefficient, 2), output_range[0])
        return min(round(to_convert/coefficient, 2), output_range[1])

def get_distance():
    # set Trigger to HIGH
    GPIO.output(TRIGGER, GPIO.HIGH)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(TRIGGER, GPIO.LOW)
```

```
    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return float(distance)

def move_motor(motor, speed, reverse=False):

    if motor == 1:
        if reverse:
            GPIO.output(IN1, GPIO.LOW)
            GPIO.output(IN2, GPIO.HIGH)
            ena_pwm.ChangeDutyCycle(speed)
        else:
            GPIO.output(IN1, GPIO.HIGH)
            GPIO.output(IN2, GPIO.LOW)
            ena_pwm.ChangeDutyCycle(speed)

    if motor == 2:
        if reverse:
            GPIO.output(IN3, GPIO.LOW)
            GPIO.output(IN4, GPIO.HIGH)
            enb_pwm.ChangeDutyCycle(speed)
        else:
            GPIO.output(IN3, GPIO.HIGH)
            GPIO.output(IN4, GPIO.LOW)
            enb_pwm.ChangeDutyCycle(speed)
```

```
def move_motors(speed, reverse=False):

    if reverse:
        GPIO.output(IN1, GPIO.LOW)
        GPIO.output(IN2, GPIO.HIGH)
        GPIO.output(IN3, GPIO.LOW)
        GPIO.output(IN4, GPIO.HIGH)
        ena_pwm.ChangeDutyCycle(speed)
        enb_pwm.ChangeDutyCycle(speed)
    else:
        GPIO.output(IN1, GPIO.HIGH)
        GPIO.output(IN2, GPIO.LOW)
        GPIO.output(IN3, GPIO.HIGH)
        GPIO.output(IN4, GPIO.LOW)
        ena_pwm.ChangeDutyCycle(speed)
        enb_pwm.ChangeDutyCycle(speed)
```

Листинг программы П.3.1. – Листинг программы основного модуля с составленными функции для нормализации значении и управления моторами и датчиков расстояния

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership
# functions
distance = ctrl.Antecedent(np.arange(0, 1, 0.01), 'distance')
motor_1 = ctrl.Consequent(np.arange(0, 1, 0.01), 'motor_1')
motor_2 = ctrl.Consequent(np.arange(0, 1, 0.01), 'motor_2')

motor_1.defuzzify_method = 'mom'
motor_2.defuzzify_method = 'mom'

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
distance['very close'] = fuzz.trapmf(distance.universe, [0, 0, 0.100, 0.150])
distance['close'] = fuzz.trapmf(
    distance.universe, [0.100, 0.150, 0.350, 0.400])
distance['normal'] = fuzz.trapmf(
    distance.universe, [0.350, 0.400, 0.600, 0.650])
distance['far'] = fuzz.trapmf(distance.universe, [0.600, 0.650, 0.850, 0.900])
```

```
distance['very far'] = fuzz.trapmf(distance.universe, [0.850, 0.900, 1, 1])

# Output membership functions
motor_1['medium'] = fuzz.gbellmf(motor_1.universe, 0.2, 2, 0)
motor_1['high'] = fuzz.gbellmf(motor_1.universe, 0.2, 2, 0.5)
motor_1['very high'] = fuzz.gbellmf(motor_1.universe, 0.2, 2, 1)

motor_2['medium'] = fuzz.gbellmf(motor_2.universe, 0.2, 2, 0)
motor_2['high'] = fuzz.gbellmf(motor_2.universe, 0.2, 2, 0.5)
motor_2['very high'] = fuzz.gbellmf(motor_2.universe, 0.2, 2, 1)

# Establish the rules for the system
rule1 = ctrl.Rule(distance['very close'],
                  (motor_1['very high'], motor_2['medium']))
rule2 = ctrl.Rule(distance['close'], (motor_1['high'], motor_2['medium']))
rule3 = ctrl.Rule(distance['normal'], (motor_1['medium'], motor_2['medium']))
rule4 = ctrl.Rule(distance['far'], (motor_1['medium'], motor_2['high']))
rule5 = ctrl.Rule(distance['very far'],
                  (motor_1['medium'], motor_2['very high']))

motion_control = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5])
motion = ctrl.ControlSystemSimulation(motion_control)

def get_fuzzy_value(distance, motor):

    motion.input['distance'] = distance
    motion.compute()

    if motor == 1:
        return float(motion.output['motor_1'])
    if motor == 2:
        return float(motion.output['motor_2'])
```

Листинг программы П.3.2. – Листинг программы расчета нечетких значении для алгоритма управления моторами платформы

```
import time
import RPi.GPIO as GPIO
from utilities import *
from MotionControl import *

old_distance = 0

if __name__ == '__main__':
    try:
        while True:

            # Get distance value from Ultra Sonic Sensor
            new_distance = get_distance()
            if new_distance > 400:
                new_distance = old_distance
            new_distance = (new_distance + old_distance)/2
            weighted_distance = round((new_distance*0.1 + old_distance*0.9), 2)
            old_distance = weighted_distance

            time.sleep(0.5)
            # Normalize it to feed into Fuzzy Logic System
            norm_dist = custom_normalize(weighted_distance, [0, 400], [0, 1])

            # Feed into FLS
            fuzz_val_mot1 = get_fuzzy_value(norm_dist, 1)
            fuzz_val_mot2 = get_fuzzy_value(norm_dist, 2)

            # Normalize output values to match up with PWM range
            norm_speed_mot1 = custom_normalize(float(fuzz_val_mot1), [0, 1], [0, 100])
            norm_speed_mot2 = custom_normalize(float(fuzz_val_mot2), [0, 1], [0, 100])

            # Move motors according to Fuzzy Logic

            print(weighted_distance, norm_speed_mot1, norm_speed_mot2)

            #move_motor(1, float(norm_speed_mot1))
            #move_motor(2, float(norm_speed_mot2))

            #time.sleep(0.01)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
```

```
        print("Process has been stopped by User")
        GPIO.cleanup()
```

Листинг программы П.3.3. – Листинг программы исполнения алгоритма управления моторами платформы

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

z_current = ctrl.Antecedent(np.arange(-1, 1, 0.01), 'z_current')
disk_voltage = ctrl.Consequent(np.arange(0, 1, 0.01), 'disk_voltage')

disk_voltage.defuzzify_method = 'mom'

# Custom membership functions can be built interactively with a familiar,
z_current['-overload'] = fuzz.trapmf(z_current.universe,
                                     [-1, -1, -0.90, -0.80])
z_current['-nominal'] = fuzz.trapmf(z_current.universe,
                                    [-0.90, -0.70, -0.20, -0.10])
z_current['turned_off'] = fuzz.trapmf(
    z_current.universe, [-0.20, -0.10, 0.10, 0.20])
z_current['+nominal'] = fuzz.trapmf(z_current.universe,
                                    [0.10, 0.20, 0.70, 0.90])
z_current['+overload'] = fuzz.trapmf(z_current.universe, [0.80, 0.90, 1, 1])

# Output membership functions
disk_voltage['OFF'] = fuzz.trimf(disk_voltage.universe, [0, 0, 0.5])
disk_voltage['ON'] = fuzz.trimf(disk_voltage.universe, [0.5, 1, 1])

rule1 = ctrl.Rule(z_current['-overload'], disk_voltage['OFF'])
rule2 = ctrl.Rule(z_current['-nominal'], disk_voltage['OFF'])
rule3 = ctrl.Rule(z_current['turned_off'], disk_voltage['OFF'])
rule4 = ctrl.Rule(z_current['+nominal'], disk_voltage['ON'])
rule5 = ctrl.Rule(z_current['+overload'], disk_voltage['OFF'])

disk_control = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5])
disk = ctrl.ControlSystemSimulation(disk_control)


def get_fuzzy_value(distance):
```

```
    disk.input['z_current'] = distance
    disk.compute()

    return float(disk.output['disk_voltage'])
```

Листинг программы П.3.4. – Листинг программы расчета нечетких значении для алгоритма управления моторами культивационного диска

```
import time
import RPi.GPIO as GPIO
from utilities import *
from DiskControl import *

old_distance = 0

if __name__ == '__main__':
    try:
        while True:

            # Get distance value from Ultra Sonic Sensor
            new_distance = get_distance()
            if new_distance > 400:
                new_distance = old_distance
            new_distance = (new_distance + old_distance)/2
            weighted_distance = round((new_distance*0.1 + old_distance*0.9), 2)
            old_distance = weighted_distance

            time.sleep(0.5)
            # Normalize it to feed into Fuzzy Logic System
            norm_dist = custom_normalize(weighted_distance, [0, 400], [0, 1])

            # Feed into FLS
            fuzz_val_mot = get_fuzzy_value(norm_dist)

            # Normalize output values to match up with PWM range
            norm_speed_mot = custom_normalize(fuzz_val_mot, [0, 1], [0, 100])

            # Move motors according to Fuzzy Logic
            move_motor(1, norm_speed_mot)

            time.sleep(0.01)
```

```
        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Process has been stopped by User")
        GPIO.cleanup()
```

Листинг программы П.3.5. – Листинг программы исполнения алгоритма управления моторами культивационного диска

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership
# functions
distance_x = ctrl.Antecedent(np.arange(-1, 1, 0.01), 'distance_x')
motor_x = ctrl.Consequent(np.arange(-1, 1, 0.01), 'motor_x')

motor_x.defuzzify_method = 'mom'

# Custom membership functions can be built interactively with a familiar,
distance_x['-far'] = fuzz.gbellmf(distance_x.universe, 0.2, 3, -1)
distance_x['-average'] = fuzz.gbellmf(distance_x.universe, 0.2, 3, -0.7)
distance_x['-close'] = fuzz.gbellmf(distance_x.universe, 0.2, 3, -0.3)
distance_x['zero'] = fuzz.trimf(distance_x.universe, [-0.3, 0, 0.3])
distance_x['+close'] = fuzz.gbellmf(distance_x.universe, 0.2, 3, 0.3)
distance_x['+average'] = fuzz.gbellmf(distance_x.universe, 0.2, 3, 0.7)
distance_x['+far'] = fuzz.gbellmf(distance_x.universe, 0.2, 3, 1)

# Output membership functions
motor_x['-high'] = fuzz.gbellmf(motor_x.universe, 0.2, 3, -1)
motor_x['-average'] = fuzz.gbellmf(motor_x.universe, 0.2, 3, -0.7)
motor_x['-low'] = fuzz.gbellmf(motor_x.universe, 0.2, 3, -0.3)
motor_x['zero'] = fuzz.trimf(motor_x.universe, [-0.3, 0, 0.3])
motor_x['+low'] = fuzz.gbellmf(motor_x.universe, 0.2, 3, 0.3)
motor_x['+average'] = fuzz.gbellmf(motor_x.universe, 0.2, 3, 0.7)
motor_x['+high'] = fuzz.gbellmf(motor_x.universe, 0.2, 3, 1)


rule1 = ctrl.Rule(distance_x['+far'], motor_x['+high'])
rule2 = ctrl.Rule(distance_x['+average'], motor_x['+average'])
rule3 = ctrl.Rule(distance_x['+close'], motor_x['+low'])
rule4 = ctrl.Rule(distance_x['zero'], motor_x['zero'])
```

```
rule5 = ctrl.Rule(distance_x['-close'], motor_x['-low'])
rule6 = ctrl.Rule(distance_x['-average'], motor_x['-average'])
rule7 = ctrl.Rule(distance_x['-far'], motor_x['-high'])

x_motion_control = ctrl.ControlSystem(
   [rule1, rule2, rule3, rule4, rule5, rule6, rule7])
x_motion = ctrl.ControlSystemSimulation(x_motion_control)

def get_fuzzy_value(distance):

   x_motion.input['distance_x'] = distance
   x_motion.compute()

   return float(x_motion.output['motor_x'])
```

Листинг программы П.3.6. – Листинг программы расчета нечетких значении для алгоритма управления приводами по координате X

```
import time
import RPi.GPIO as GPIO
from utilities import *
from XMotionControl import *

old_distance = 0

if __name__ == '__main__':
   try:
      while True:

         # Get distance value from Ultra Sonic Sensor
         new_distance = get_distance()
         if new_distance > 400:
            new_distance = old_distance
         new_distance = (new_distance + old_distance)/2
         weighted_distance = round((new_distance*0.1 + old_distance*0.9), 2)
         old_distance = weighted_distance

         time.sleep(0.5)
         # Normalize it to feed into Fuzzy Logic System
         norm_dist = custom_normalize(weighted_distance, [0, 400], [-1, 1])

         # Feed into FLS
```

```
            fuzz_val_mot = get_fuzzy_value(norm_dist)

            # Normalize output values to match up with PWM range
            norm_speed_mot = custom_normalize(fuzz_val_mot, [-1, 1], [0, 100])

            # Move motors according to Fuzzy Logic
            if fuzz_val_mot < 0:
               move_motor(1, norm_speed_mot, reverse=True)
            else:
               move_motor(1, norm_speed_mot)

            time.sleep(0.01)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Process has been stopped by User")
        GPIO.cleanup()
```

Листинг программы П.3.7. – Листинг программы исполнения алгоритма управления приводами по координате X

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership
# functions
distance_y = ctrl.Antecedent(np.arange(-1, 1, 0.01), 'distance_y')
motor_y = ctrl.Consequent(np.arange(-1, 1, 0.01), 'motor_y')

motor_y.defuzzify_method = 'mom'

# Custom membership functions can be built interactively with a familiar,
distance_y['-far'] = fuzz.gbellmf(distance_y.universe, 0.2, 3, -1)
distance_y['-average'] = fuzz.gbellmf(distance_y.universe, 0.2, 3, -0.7)
distance_y['-close'] = fuzz.gbellmf(distance_y.universe, 0.2, 3, -0.3)
distance_y['zero'] = fuzz.trimf(distance_y.universe, [-0.3, 0, 0.3])
distance_y['+close'] = fuzz.gbellmf(distance_y.universe, 0.2, 3, 0.3)
distance_y['+average'] = fuzz.gbellmf(distance_y.universe, 0.2, 3, 0.7)
distance_y['+far'] = fuzz.gbellmf(distance_y.universe, 0.2, 3, 1)

# Output membership functions
```

```
motor_y['-high'] = fuzz.gbellmf(motor_y.universe, 0.2, 3, -1)
motor_y['-average'] = fuzz.gbellmf(motor_y.universe, 0.2, 3, -0.7)
motor_y['-low'] = fuzz.gbellmf(motor_y.universe, 0.2, 3, -0.3)
motor_y['zero'] = fuzz.trimf(motor_y.universe, [-0.3, 0, 0.3])
motor_y['+low'] = fuzz.gbellmf(motor_y.universe, 0.2, 3, 0.3)
motor_y['+average'] = fuzz.gbellmf(motor_y.universe, 0.2, 3, 0.7)
motor_y['+high'] = fuzz.gbellmf(motor_y.universe, 0.2, 3, 1)

rule1 = ctrl.Rule(distance_y['+far'], motor_y['+high'])
rule2 = ctrl.Rule(distance_y['+average'], motor_y['+average'])
rule3 = ctrl.Rule(distance_y['+close'], motor_y['+low'])
rule4 = ctrl.Rule(distance_y['zero'], motor_y['zero'])
rule5 = ctrl.Rule(distance_y['-close'], motor_y['-low'])
rule6 = ctrl.Rule(distance_y['-average'], motor_y['-average'])
rule7 = ctrl.Rule(distance_y['-far'], motor_y['-high'])

y_motion_control = ctrl.ControlSystem(
    [rule1, rule2, rule3, rule4, rule5, rule6, rule7])
y_motion = ctrl.ControlSystemSimulation(y_motion_control)

def get_fuzzy_value(distance):

    y_motion.input['distance_y'] = distance
    y_motion.compute()

    return float(y_motion.output['motor_y'])
```

Листинг программы П.3.8. – Листинг программы расчета нечетких значении для алгоритма управления приводами по координате У

```
import time
import RPi.GPIO as GPIO
from utilities import *
from YMotionControl import *

old_distance = 0

if __name__ == '__main__':
    try:
        while True:
```

```
            # Get distance value from Ultra Sonic Sensor
            new_distance = get_distance()
            if new_distance > 400:
                new_distance = old_distance
            new_distance = (new_distance + old_distance)/2
            weighted_distance = round((new_distance*0.1 + old_distance*0.9), 2)
            old_distance = weighted_distance

            time.sleep(0.5)
            # Normalize it to feed into Fuzzy Logic System
            norm_dist = custom_normalize(weighted_distance, [0, 400], [-1, 1])

            # Feed into FLS
            fuzz_val_mot = get_fuzzy_value(norm_dist)

            # Normalize output values to match up with PWM range
            norm_speed_mot = custom_normalize(fuzz_val_mot, [-1, 1], [0, 100])

            # Move motors according to Fuzzy Logic
            if fuzz_val_mot < 0:
                move_motor(1, norm_speed_mot, reverse=True)
            else:
                move_motor(1, norm_speed_mot)

            time.sleep(0.01)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Process has been stopped by User")
        GPIO.cleanup()
```

Листинг программы П.3.9. – Листинг программы исполнения алгоритма управления приводами по координате У

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership
# functions
current_x = ctrl.Antecedent(np.arange(-1, 1, 0.01), 'current_x')
current_y = ctrl.Antecedent(np.arange(-1, 1, 0.01), 'current_y')
```

```
current_z = ctrl.Antecedent(np.arange(-2, 2, 0.01), 'current_z')
motor_z = ctrl.Consequent(np.arange(-1, 1, 0.01), 'motor_z')

motor_z.defuzzify_method = 'mom'

# Custom membership functions can be built interactively with a familiar,
current_x['+present'] = fuzz.trapmf(current_x.universe, [-1, -1, -0.2, -0.1])
current_x['turned_off'] = fuzz.trimf(current_x.universe, [-0.2, 0, 0.2])
current_x['-present'] = fuzz.trapmf(current_x.universe, [0.1, 0.2, 1, 1])

current_y['+present'] = fuzz.trapmf(current_y.universe, [-1, -1, -0.2, -0.1])
current_y['turned_off'] = fuzz.trimf(current_y.universe, [-0.2, 0, 0.2])
current_y['-present'] = fuzz.trapmf(current_y.universe, [0.1, 0.2, 1, 1])

current_z['-overload'] = fuzz.trapmf(current_z.universe, [-2, -2, -1.7, -1.5])
current_z['-nominal'] = fuzz.trapmf(current_z.universe,
                                    [-1.7, -1.5, -0.5, -0.1])
current_z['turned_off'] = fuzz.trimf(current_z.universe, [-0.25, 0, 0.25])
current_z['+nominal'] = fuzz.trapmf(current_z.universe, [0.1, 0.5, 1.5, 1.7])
current_z['+overload'] = fuzz.trapmf(current_z.universe, [1.5, 1.7, 2, 2])

# Output membership functions
motor_z['up'] = fuzz.trimf(motor_z.universe, [-1, -1, -0.3])
motor_z['set'] = fuzz.trimf(motor_z.universe, [-0.3, -0.2, -0.1])
motor_z['off'] = fuzz.trimf(motor_z.universe, [-0.1, 0, 0.1])
motor_z['down'] = fuzz.trimf(motor_z.universe, [0.1, 1, 1])

# Establish the rules for the system
rule1 = ctrl.Rule(current_x["-present"] & current_y["-present"]
                  & current_z["turned_off"], motor_z["off"])
rule2 = ctrl.Rule(current_x["-present"] & current_y["+present"]
                  & current_z["turned_off"], motor_z["off"])
rule3 = ctrl.Rule(current_x["-present"] & current_y["turned_off"]
                  & current_z["turned_off"], motor_z["off"])
rule4 = ctrl.Rule(current_x["+present"] & current_y["-present"]
                  & current_z["turned_off"], motor_z["off"])
rule5 = ctrl.Rule(current_x["+present"] & current_y["turned_off"]
                  & current_z["turned_off"], motor_z["off"])
rule6 = ctrl.Rule(current_x["+present"] & current_y["+present"]
                  & current_z["turned_off"], motor_z["off"])
rule7 = ctrl.Rule(current_x["turned_off"] & current_y["turned_off"]
                  & current_z["turned_off"], motor_z["down"])
```

```
rule8 = ctrl.Rule(current_x["turned_off"] & current_y["turned_off"]
            & current_z["+overload"], motor_z["up"])
rule9 = ctrl.Rule(current_x["turned_off"] & current_y["turned_off"]
            & current_z["+nominal"], motor_z["down"])
rule10 = ctrl.Rule(current_x["turned_off"] & current_y["turned_off"]
             & current_z["-nominal"], motor_z["up"])
rule11 = ctrl.Rule(current_x["turned_off"] & current_y["turned_off"]
             & current_z["-overload"], motor_z["set"])

z_motion_control = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5,
                                       rule6, rule7, rule8, rule9, rule10,
                                       rule11])
z_motion = ctrl.ControlSystemSimulation(z_motion_control)

def get_fuzzy_value(current_x, current_y, current_z):

    z_motion.input['current_x'] = current_x
    z_motion.input['current_y'] = current_y
    z_motion.input['current_z'] = current_z
    # Crunch the numbers
    z_motion.compute()

    return float(z_motion.output['motor_z'])
```

Листинг программы П.3.10. – Листинг программы расчета нечетких значении для алгоритма управления приводами по координате Z

```
import time
import RPi.GPIO as GPIO
from utilities import *
from ZMotionControl import *

old_distance = 0

if __name__ == '__main__':
    try:
        while True:

            # Get distance value from Ultra Sonic Sensor
            new_distance = get_distance()
            if new_distance > 400:
                new_distance = old_distance
```

```
        new_distance = (new_distance + old_distance)/2
        weighted_distance = round((new_distance*0.1 + old_distance*0.9), 2)
        old_distance = weighted_distance

        time.sleep(0.5)
        # Normalize it to feed into Fuzzy Logic System
        norm_dist1 = custom_normalize(weighted_distance, [0, 400], [-1, 1])
        norm_dist2 = custom_normalize(weighted_distance, [0, 400], [-2, 2])

        # Feed into FLS
        fuzz_val_mot = get_fuzzy_value(norm_dist1, norm_dist1, norm_dist2)

        # Normalize output values to match up with PWM range
        norm_speed_mot = custom_normalize(fuzz_val_mot, [-1, 1], [0, 100])

        # Move motors according to Fuzzy Logic
        if fuzz_val_mot < 0:
            move_motor(1, norm_speed_mot, reverse=True)
        else:
            move_motor(1, norm_speed_mot)

        time.sleep(0.01)

    # Reset by pressing CTRL + C
except KeyboardInterrupt:
    print("Process has been stopped by User")
    GPIO.cleanup()
```

Листинг программы П.3.11. – Листинг программы исполнения алгоритма управления приводами по координате Z

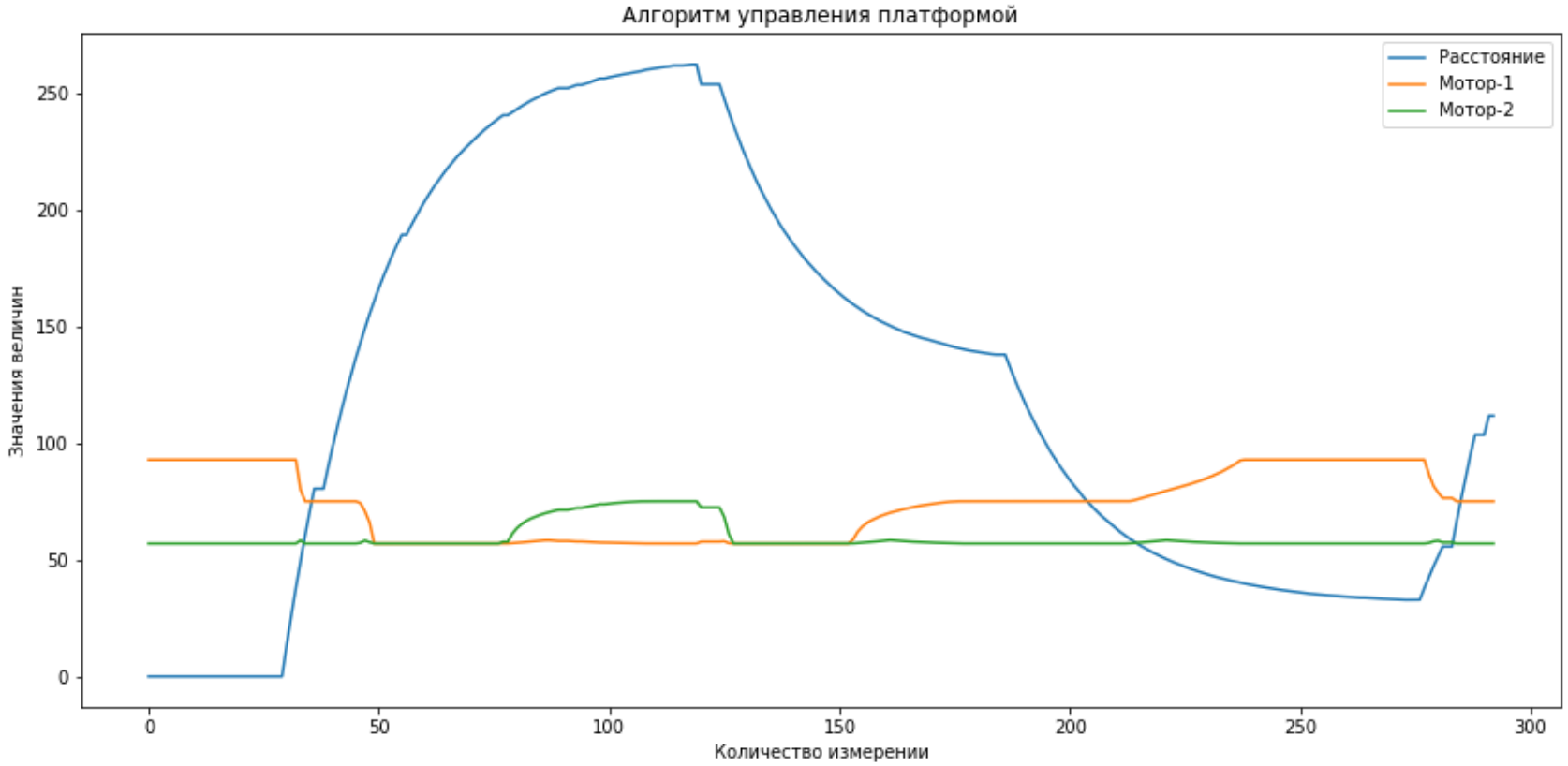

Рисунок П.3.12. – Результаты экспериментальных тестировании алгоритма управления мобильной платформой

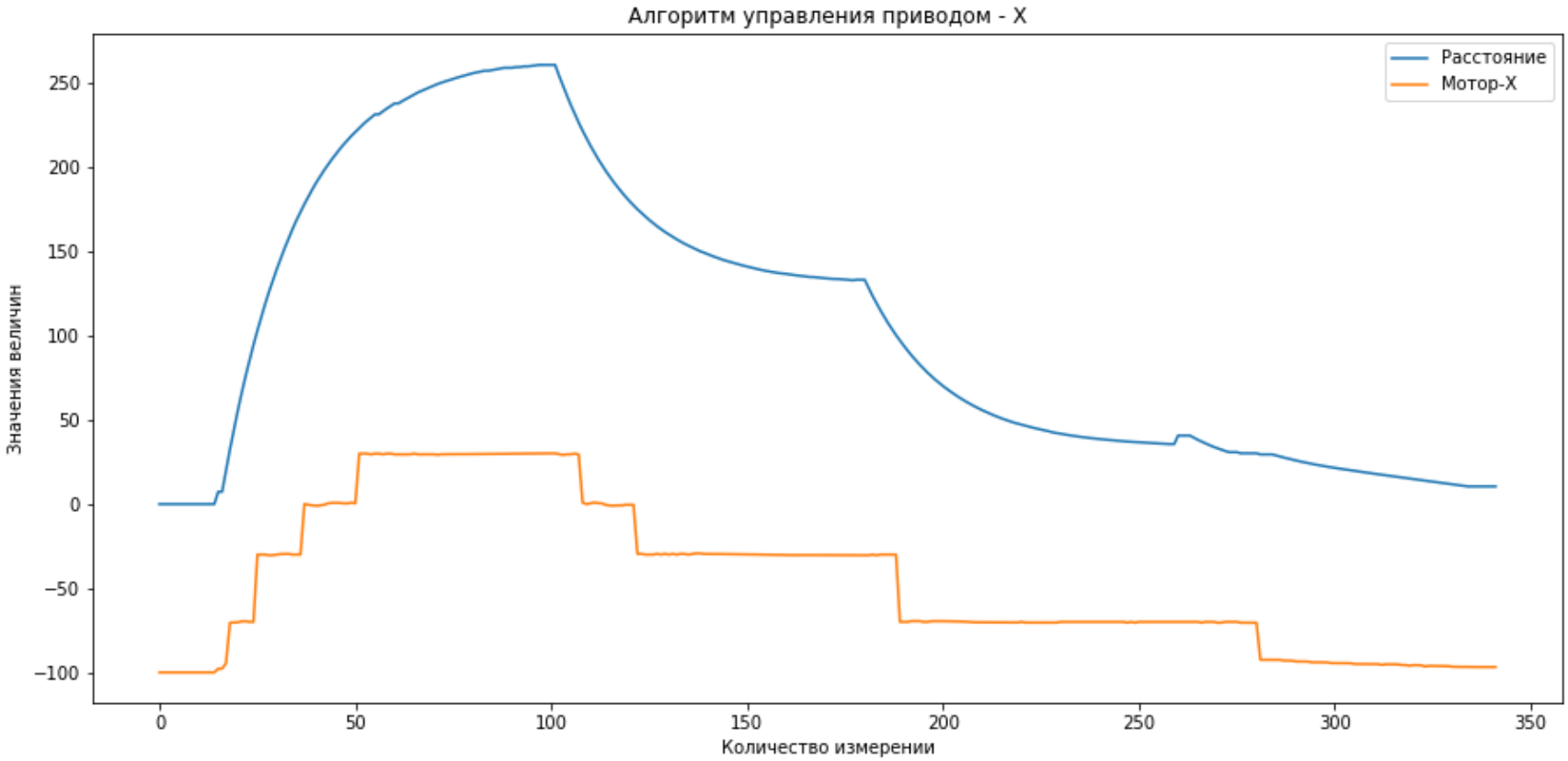

Рисунок П.3.13. – Результаты экспериментальных тестировании алгоритма управления приводом координаты X

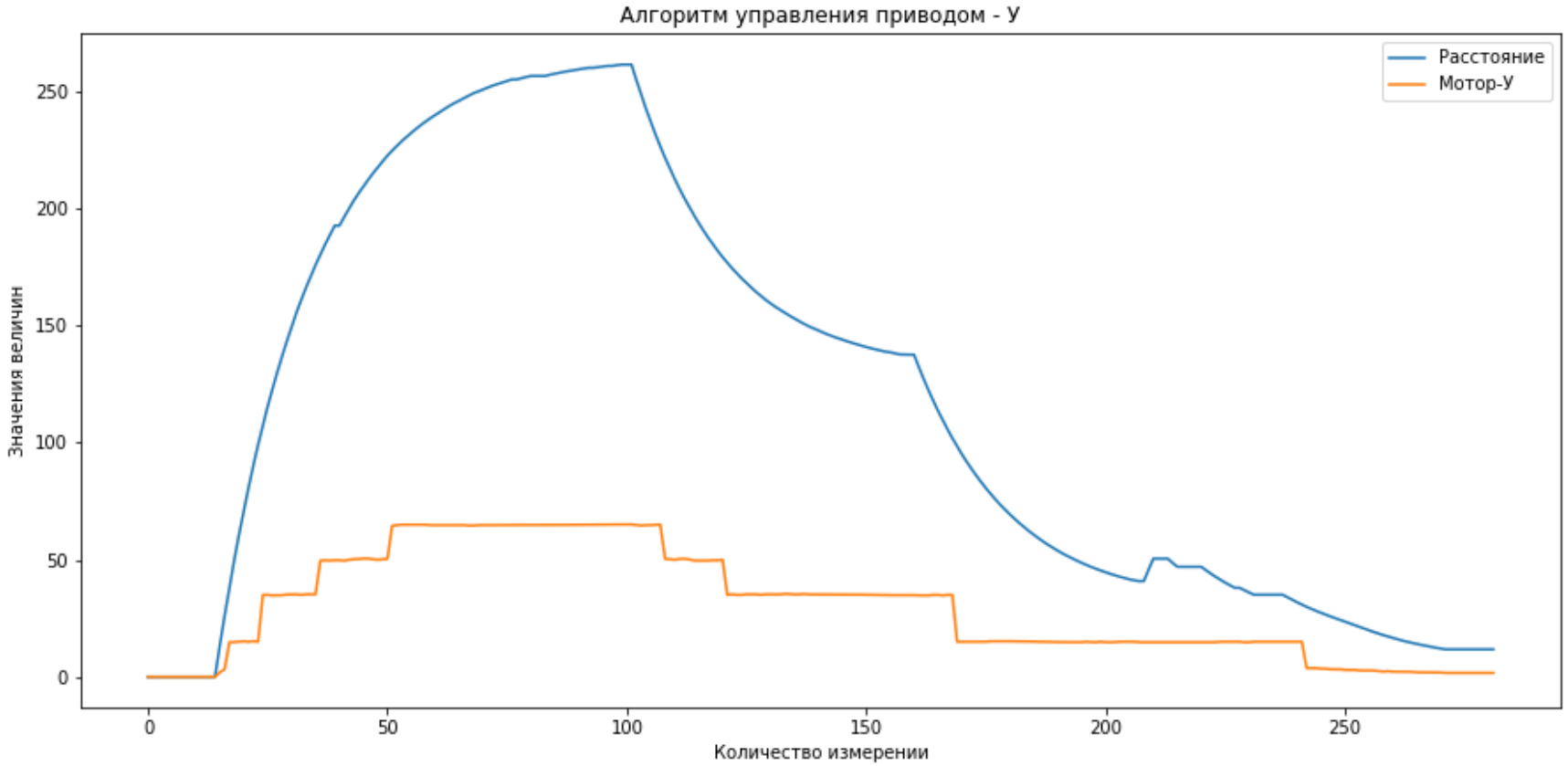

Рисунок П.3.14. – Результаты экспериментальных тестировании алгоритма управления приводом координаты У

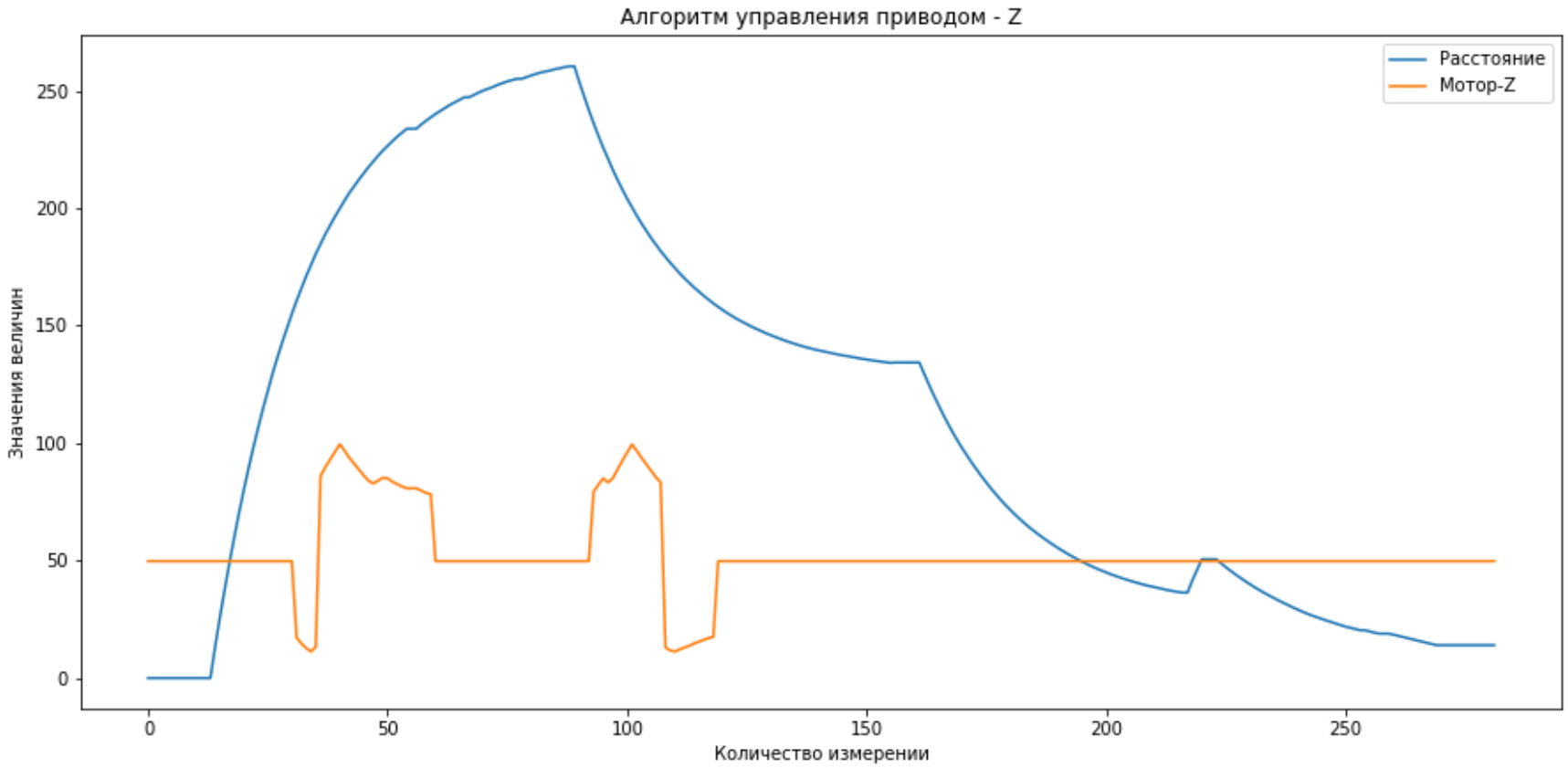

Рисунок П.3.15. – Результаты экспериментальных тестировании алгоритма управления приводом координаты Z

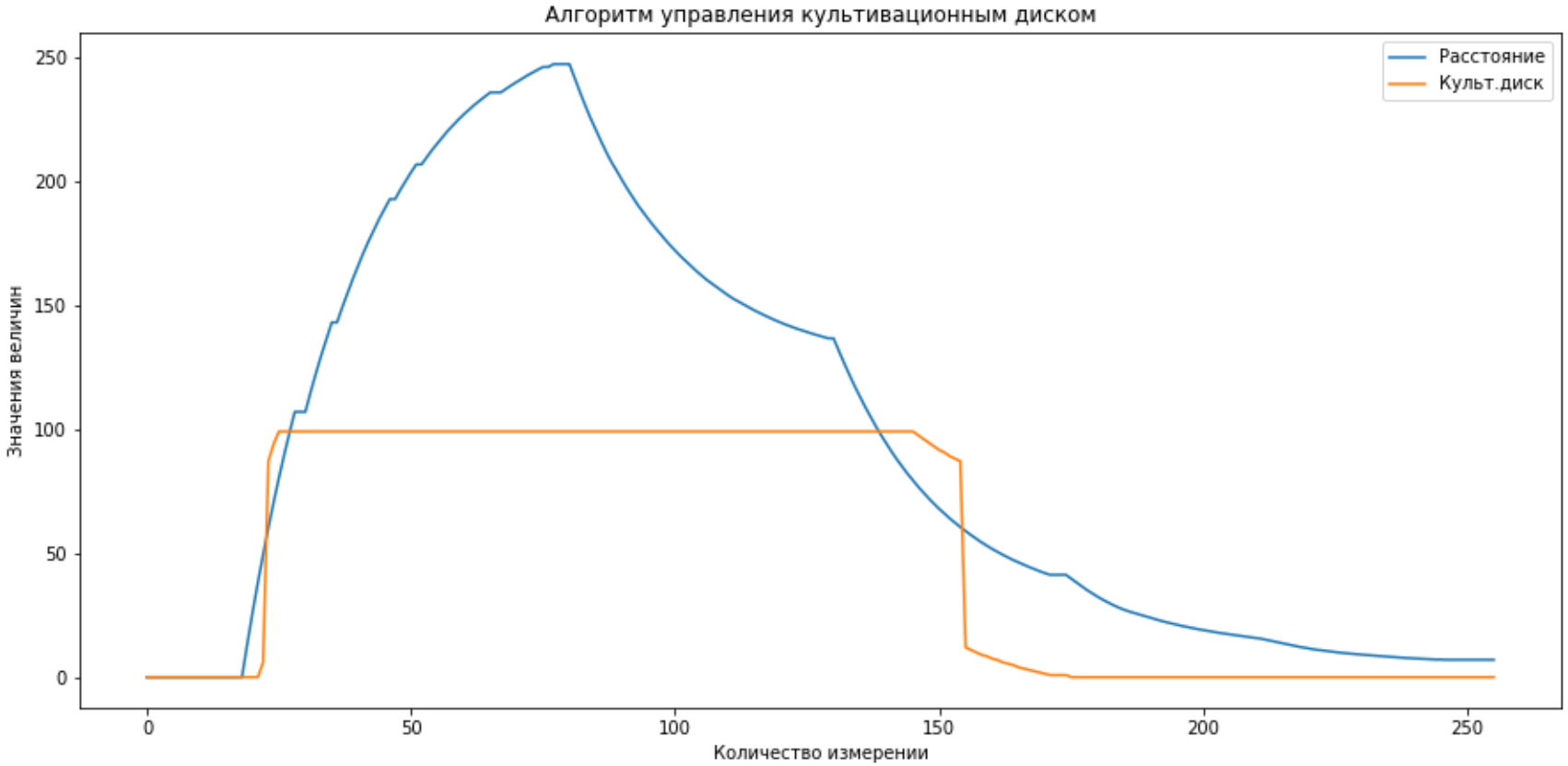

Рисунок П.3.16. – Результаты экспериментальных тестировании алгоритма управления приводом культивационного диска