ROBOTICS TECHNICAL REPORT

Robot Joint Using IRB4600 & UR3e in Webots

Ditujukan Untuk Memenuhi Tugas Mata Kuliah Robotika

Disusun Oleh :

Fahmi Nanda Saputro - 1103200203

UNIVERSITAS TELKOM

BANDUNG

2023

BAB I

INTRODUCTIONS

1.1 ABB IRB4600

Robot IRB 4600 adalah sebuah robot dengan performa kelas atas yang biasa dipakai di perusahaan besar karena kehandalannya. Robot ini terdapat 6 sistem *Joints* yang memungkinkan tugas - tugas yang berat dan kompleks yang membutuhkan presisi dan efisiensi. Robot ini mampu mengangkat beban hingga 40kg dengan jarak jangkauan 2,55 meter. Sehingga, robot ini mampu mengangkat beban berat dan besar. Untuk industri robot ini biasanya digunakan di manufaktur otomotif, elektronik, dan logistik.

1.2 UR3e Robots

Universal Robots UR3e adalah sebuah robot kolaboratif yang dirancang untuk bekerja sama operator. Bentuk yang ringan dan kecil, serta kemudahan dalam penggunaan membuat robot ini aman berinteraksi dengan manusia. UR3e menawarkan juga beberapa *joints* yang fleksibel dan mampu mengankat beban dengan kapasitas hingga 3 kg, sehingga penggunanya sangat berguna untuk tugas yang halus dan presisi. UR3e biasa digunakan dalam industri elektronik, medis, dan manufaktur skala kecil.

1.3 Fitur - fitur

Kedua Robot ini memiliki fitur - fitur yang canggih yang bisa meningkatkan kinerja seperti :

1.3.1 Sistem Kontrol Lanjutan

Kedua robot ini memiliki sistem kontrol yang canggih, memungkinkan pergerakan presisi dan mulus. banyak sekali mode kontrol seperti, kontrol posisi, kecepatan, dan torsi, yang mampu bekerja dibawah arahan tertentu.

1.3.2 Fitur Keselamatan

Banyak sekali fitur keselamatan dari kedua robot ini seperti, sensor pendeteksi tabrakan dan mekanisme keselamatan berbasis perangkat lunak.

1.3.3 Antarmuka Pengguna yang Mudah Digunakan

Antarmuka pemrograman yang disediakan oleh ABB dan Universal Robots dirancang agar mudah dipahami dan digunakan. Sehingga, memungkinkan operator dan insinyur untuk dengan mudah memprogram dan mengendalikan robot tanpa keahlian robotika yang luas. Aksesibilitas ini memungkinkan implementasi yang cepat dan integrasi yang efisien ke dalam alur kerja yang sudah ada.

BAB II

JOINTS CONTROL in WEBOTS

2.1 Penjelasan

Dalam aplikasi Webots terdapat beberapa tools yang dapat mengatur dan mengontrol dari robot dan bisa membuat sebuah simulasi dari robot yang telah dibuat. Serta bagaimana antar *joints* bisa saling berkesinambungan satu sama lain. Saat user menggunakan aplikasi Webots, bisa memilih dan mengatur dari *joints property* seperti dari *Position, Velocity, and Torque limits* dan nantinya bisa diatur dan disimulasikan antar *joints*. Pengaturan *Joints* dapat diambil dari beberapa metode, termasuk *direct position control, PID Control, Trajectory Planning Algorithms*. Dalam aplikasi Webots, pengguna dapat mengakses dan memodifikasi *Joints Parameter*, *Set Joint Position or Velocity,* dan *Retrieve Joint feedback data.*

BAB III

CONCLUSION

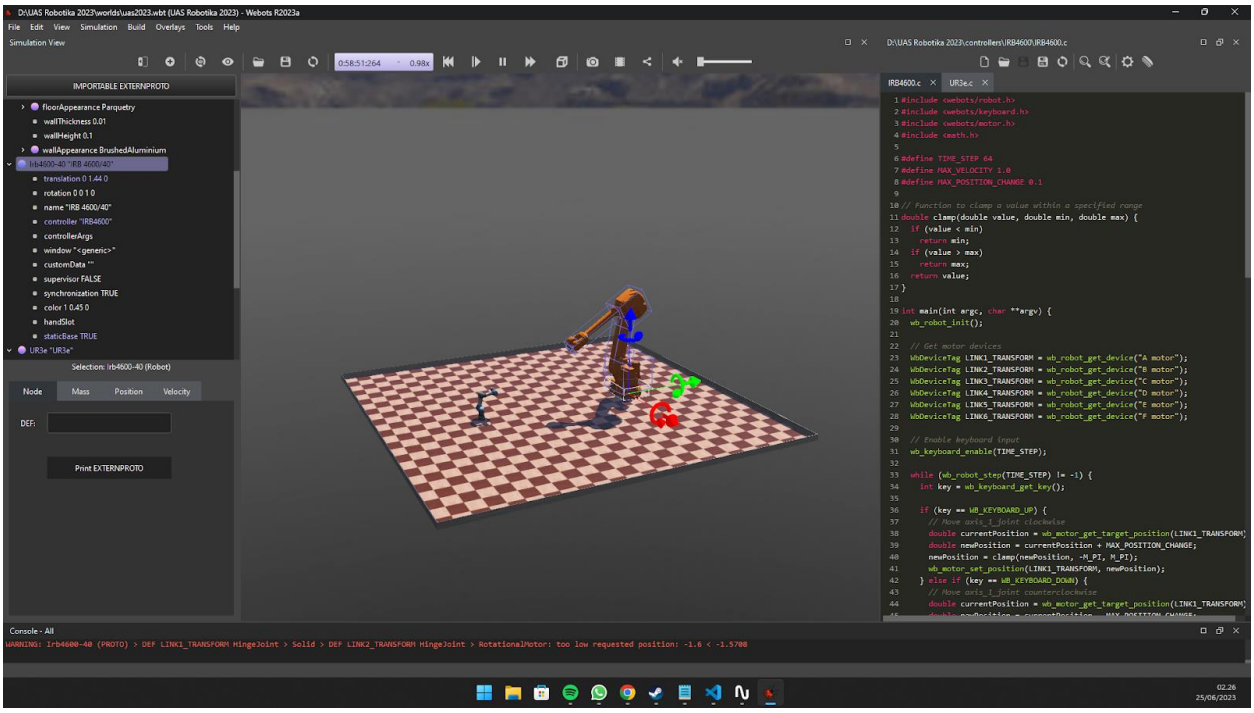
3.1 Penutup

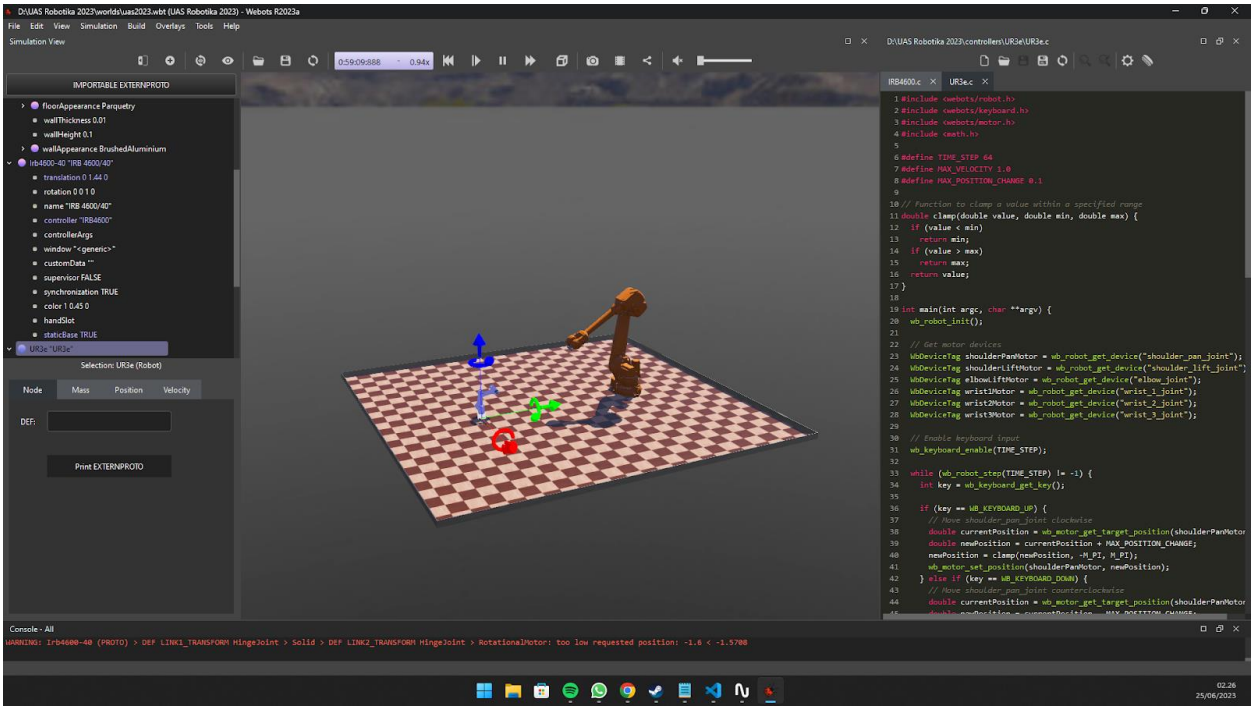
Secara keseluruhan, integrasi joint robot menggunakan model IRB 4600 dan UR3e di lingkungan simulasi Webots menawarkan platform yang kuat untuk mempelajari dan mengimplementasikan aplikasi robotika. Kontrol yang tepat dan perilaku yang realistis dari joint memungkinkan para peneliti, insinyur, dan pendidik untuk menjelajahi berbagai strategi kontrol, menguji algoritma, dan mengevaluasi kinerja sistem robotik dalam lingkungan virtual. Kombinasi antara Webots dan model robot ini membuka peluang untuk inovasi dan kemajuan dalam bidang robotika.

BAB IV

DOCUMENTATION

4.1 Aplikasi Webots





4.2 Program Robot

Untuk program yang digunakan dalam aplikasi webots adalah bahasa C

4.2.1 Program untuk IRB 4600

```
#include <webots/robot.h>
#include <webots/keyboard.h>
#include <webots/motor.h>
#include <math.h>

#define TIME_STEP 64
#define MAX_VELOCITY 1.0
#define MAX_POSITION_CHANGE 0.1

// Function to clamp a value within a specified range
double clamp(double value, double min, double max) {
  if (value < min)
    return min;
  if (value > max)
    return max;
  return value;
}

int main(int argc, char **argv) {
  wb_robot_init();

  // Get motor devices
  WbDeviceTag LINK1_TRANSFORM = wb_robot_get_device("A motor");
  WbDeviceTag LINK2_TRANSFORM = wb_robot_get_device("B motor");
  WbDeviceTag LINK3_TRANSFORM = wb_robot_get_device("C motor");
  WbDeviceTag LINK4_TRANSFORM = wb_robot_get_device("D motor");
  WbDeviceTag LINK5_TRANSFORM = wb_robot_get_device("E motor");
  WbDeviceTag LINK6_TRANSFORM = wb_robot_get_device("F motor");

  // Enable keyboard input
  wb_keyboard_enable(TIME_STEP);

  while (wb_robot_step(TIME_STEP) != -1) {
    int key = wb_keyboard_get_key();

    if (key == WB_KEYBOARD_UP) {
      // Move axis_1_joint clockwise
      double currentPosition =
wb_motor_get_target_position(LINK1_TRANSFORM);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK1_TRANSFORM, newPosition);
    } else if (key == WB_KEYBOARD_DOWN) {
      // Move axis_1_joint counterclockwise
      double currentPosition =
wb_motor_get_target_position(LINK1_TRANSFORM);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
```

```
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK1_TRANSFORM, newPosition);
    } else if (key == WB_KEYBOARD_LEFT) {
      // Move axis_2_joint clockwise
      double currentPosition =
wb_motor_get_target_position(LINK2_TRANSFORM);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK2_TRANSFORM, newPosition);
    } else if (key == WB_KEYBOARD_RIGHT) {
      // Move axis_2_joint counterclockwise
      double currentPosition =
wb_motor_get_target_position(LINK2_TRANSFORM);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK2_TRANSFORM, newPosition);
    } else if (key == 'A' || key == 'a') {
      // Move axis_3_joint clockwise
      double currentPosition =
wb_motor_get_target_position(LINK3_TRANSFORM);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK3_TRANSFORM, newPosition);
    } else if (key == 'D' || key == 'd') {
      // Move axis_3_joint counterclockwise
      double currentPosition =
wb_motor_get_target_position(LINK3_TRANSFORM);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK3_TRANSFORM, newPosition);
    } else if (key == 'Q' || key == 'q') {
      // Move axis_4_joint clockwise
      double currentPosition =
wb_motor_get_target_position(LINK4_TRANSFORM);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK4_TRANSFORM, newPosition);
    } else if (key == 'E' || key == 'e') {
      // Move axis_4_joint counterclockwise
      double currentPosition =
wb_motor_get_target_position(LINK4_TRANSFORM);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK4_TRANSFORM, newPosition);
    } else if (key == 'Z' || key == 'z') {
      // Move axis_5_joint clockwise
      double currentPosition =
wb_motor_get_target_position(LINK5_TRANSFORM);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK5_TRANSFORM, newPosition);
    } else if (key == 'C' || key == 'c') {
      // Move axis_5_joint counterclockwise
```

```
      double currentPosition =
wb_motor_get_target_position(LINK5_TRANSFORM);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK5_TRANSFORM, newPosition);
    } else if (key == 'X' || key == 'x') {
      // Move axis_6_joint clockwise
      double currentPosition =
wb_motor_get_target_position(LINK6_TRANSFORM);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK6_TRANSFORM, newPosition);
    } else if (key == 'V' || key == 'v') {
      // Move axis_6_joint counterclockwise
      double currentPosition =
wb_motor_get_target_position(LINK6_TRANSFORM);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(LINK6_TRANSFORM, newPosition);
    } else if (key == WB_KEYBOARD_ALT) {
      // Stop all joints
      wb_motor_set_velocity(LINK1_TRANSFORM, 0.0);
      wb_motor_set_velocity(LINK2_TRANSFORM, 0.0);
      wb_motor_set_velocity(LINK3_TRANSFORM, 0.0);
      wb_motor_set_velocity(LINK4_TRANSFORM, 0.0);
      wb_motor_set_velocity(LINK5_TRANSFORM, 0.0);
      wb_motor_set_velocity(LINK6_TRANSFORM, 0.0);
    }
  }

  wb_robot_cleanup();

  return 0;
}
```

Penjelasan :

1. **#include <webots/robot.h>**: Memasukkan pustaka "robot.h" yang menyediakan fungsi-fungsi untuk menginisialisasi dan membersihkan lingkungan simulasi robot Webots.
2. **#include <webots/keyboard.h>**: Memasukkan pustaka "keyboard.h" yang menyediakan fungsi-fungsi untuk mengakses input keyboard.
3. **#include <webots/motor.h>**: Memasukkan pustaka "motor.h" yang menyediakan fungsi-fungsi untuk mengontrol motor atau sendi pada robot Webots.
4. **#include <math.h>**: Memasukkan pustaka "math.h" yang menyediakan fungsi-fungsi matematika seperti fungsi trigonometri.
5. **#define TIME_STEP 64**: Mendefinisikan konstanta **TIME_STEP** dengan nilai 64, yang merupakan interval waktu dalam milidetik antara setiap langkah simulasi.
6. **#define MAX_VELOCITY 1.0**: Mendefinisikan konstanta **MAX_VELOCITY** dengan nilai 1.0, yang merupakan kecepatan maksimum yang diizinkan untuk pergerakan sendi.

7. **#define MAX_POSITION_CHANGE 0.1**: Mendefinisikan konstanta **MAX_POSITION_CHANGE** dengan nilai 0.1, yang merupakan perubahan posisi maksimum yang diizinkan untuk setiap langkah.
8. **double clamp(double value, double min, double max)**: Mendefinisikan fungsi **clamp()** yang mengambil tiga argumen: **value** (nilai yang akan dibatasi), **min** (nilai minimum yang diizinkan), dan **max** (nilai maksimum yang diizinkan). Fungsi ini digunakan untuk membatasi suatu nilai dalam rentang tertentu dan mengembalikan nilai yang telah dibatasi.
9. **int main(int argc, char **argv)**: Memulai fungsi **main()** sebagai titik masuk utama program. Fungsi ini mengambil argumen **argc** dan **argv**, meskipun tidak digunakan dalam kode ini.
10. **wb_robot_init()**: Menginisialisasi lingkungan simulasi Webots.
11. **WbDeviceTag LINK1_TRANSFORM = wb_robot_get_device("A motor");**: Mendapatkan tag perangkat (device) motor untuk sendi "A motor" pada robot. Perangkat ini akan digunakan untuk mengontrol pergerakan sendi tersebut.
12. **WbDeviceTag LINK2_TRANSFORM = wb_robot_get_device("B motor");**: Mendapatkan tag perangkat motor untuk sendi "B motor" pada robot.
13. **WbDeviceTag LINK3_TRANSFORM = wb_robot_get_device("C motor");**: Mendapatkan tag perangkat motor untuk sendi "C motor" pada robot.
14. **WbDeviceTag LINK4_TRANSFORM = wb_robot_get_device("D motor");**: Mendapatkan tag perangkat motor untuk sendi "D motor" pada robot.
15. **WbDeviceTag LINK5_TRANSFORM = wb_robot_get_device("E motor");**: Mendapatkan tag perangkat motor untuk sendi "E motor" pada robot.
16. **WbDeviceTag LINK6_TRANSFORM = wb_robot_get_device("F motor");**: Mendapatkan tag perangkat motor untuk sendi "F motor" pada robot.
17. **wb_keyboard_enable(TIME_STEP)**: Mengaktifkan input keyboard dengan interval **TIME_STEP** antara pembacaan input.
18. **while (wb_robot_step(TIME_STEP) != -1) { ... }**: Memulai loop while yang berjalan selama simulasi berjalan. Setiap iterasi dalam loop ini mewakili langkah simulasi sebesar **TIME_STEP** milidetik. Loop akan berhenti jika **wb_robot_step()** mengembalikan nilai -1, menunjukkan bahwa simulasi telah berakhir.
19. **int key = wb_keyboard_get_key();**: Membaca input dari keyboard dan menyimpannya dalam variabel **key** yang menunjukkan tombol apa yang ditekan.
20. Sekumpulan pernyataan **if-else** digunakan untuk mengatur pergerakan sendi berdasarkan input keyboard yang diterima. Setiap pernyataan **if** memeriksa tombol yang ditekan menggunakan variabel **key** dan melakukan perubahan posisi sendi yang sesuai menggunakan fungsi-fungsi Webots seperti **wb_motor_get_target_position()** dan **wb_motor_set_position()**. Posisi baru sendi dijaga agar tetap berada dalam rentang yang diizinkan menggunakan fungsi **clamp()**.
21. **wb_robot_cleanup()**: Membersihkan lingkungan simulasi Webots setelah simulasi selesai.
22. **return 0;**: Mengembalikan nilai 0 sebagai kode status program yang menandakan bahwa program berakhir tanpa kesalahan.

4.2.2 Program untuk UR3e

```
#include <webots/robot.h>
```

```c
#include <webots/keyboard.h>
#include <webots/motor.h>
#include <math.h>

#define TIME_STEP 64
#define MAX_VELOCITY 1.0
#define MAX_POSITION_CHANGE 0.1

// Function to clamp a value within a specified range
double clamp(double value, double min, double max) {
  if (value < min)
    return min;
  if (value > max)
    return max;
  return value;
}

int main(int argc, char **argv) {
  wb_robot_init();

  // Get motor devices
  WbDeviceTag shoulderPanMotor =
wb_robot_get_device("shoulder_pan_joint");
  WbDeviceTag shoulderLiftMotor =
wb_robot_get_device("shoulder_lift_joint");
  WbDeviceTag elbowLiftMotor = wb_robot_get_device("elbow_joint");
  WbDeviceTag wrist1Motor = wb_robot_get_device("wrist_1_joint");
  WbDeviceTag wrist2Motor = wb_robot_get_device("wrist_2_joint");
  WbDeviceTag wrist3Motor = wb_robot_get_device("wrist_3_joint");

  // Enable keyboard input
  wb_keyboard_enable(TIME_STEP);

  while (wb_robot_step(TIME_STEP) != -1) {
    int key = wb_keyboard_get_key();

    if (key == WB_KEYBOARD_UP) {
      // Move shoulder_pan_joint clockwise
      double currentPosition =
wb_motor_get_target_position(shoulderPanMotor);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(shoulderPanMotor, newPosition);
    } else if (key == WB_KEYBOARD_DOWN) {
      // Move shoulder_pan_joint counterclockwise
      double currentPosition =
wb_motor_get_target_position(shoulderPanMotor);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(shoulderPanMotor, newPosition);
    } else if (key == WB_KEYBOARD_LEFT) {
      // Move shoulder_lift_joint clockwise
```

```
      double currentPosition =
wb_motor_get_target_position(shoulderLiftMotor);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(shoulderLiftMotor, newPosition);
    } else if (key == WB_KEYBOARD_RIGHT) {
      // Move shoulder_lift_joint counterclockwise
      double currentPosition =
wb_motor_get_target_position(shoulderLiftMotor);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(shoulderLiftMotor, newPosition);
    } else if (key == 'A' || key == 'a') {
      // Move elbow_joint clockwise
      double currentPosition =
wb_motor_get_target_position(elbowLiftMotor);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(elbowLiftMotor, newPosition);
    } else if (key == 'D' || key == 'd') {
      // Move elbow_joint counterclockwise
      double currentPosition =
wb_motor_get_target_position(elbowLiftMotor);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(elbowLiftMotor, newPosition);
    } else if (key == 'Q' || key == 'q') {
      // Move wrist_1_joint clockwise
      double currentPosition = wb_motor_get_target_position(wrist1Motor);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(wrist1Motor, newPosition);
    } else if (key == 'E' || key == 'e') {
      // Move wrist_1_joint counterclockwise
      double currentPosition = wb_motor_get_target_position(wrist1Motor);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(wrist1Motor, newPosition);
    } else if (key == 'Z' || key == 'z') {
      // Move wrist_2_joint clockwise
      double currentPosition = wb_motor_get_target_position(wrist2Motor);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(wrist2Motor, newPosition);
    } else if (key == 'C' || key == 'c') {
      // Move wrist_2_joint counterclockwise
      double currentPosition = wb_motor_get_target_position(wrist2Motor);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(wrist2Motor, newPosition);
    } else if (key == 'X' || key == 'x') {
      // Move wrist_3_joint clockwise
      double currentPosition = wb_motor_get_target_position(wrist3Motor);
      double newPosition = currentPosition + MAX_POSITION_CHANGE;
```

```
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(wrist3Motor, newPosition);
    } else if (key == 'V' || key == 'v') {
      // Move wrist_3_joint counterclockwise
      double currentPosition = wb_motor_get_target_position(wrist3Motor);
      double newPosition = currentPosition - MAX_POSITION_CHANGE;
      newPosition = clamp(newPosition, -M_PI, M_PI);
      wb_motor_set_position(wrist3Motor, newPosition);
    } else if (key == WB_KEYBOARD_ALT) {
      // Stop all joints
      wb_motor_set_velocity(shoulderPanMotor, 0.0);
      wb_motor_set_velocity(shoulderLiftMotor, 0.0);
      wb_motor_set_velocity(elbowLiftMotor, 0.0);
      wb_motor_set_velocity(wrist1Motor, 0.0);
      wb_motor_set_velocity(wrist2Motor, 0.0);
      wb_motor_set_velocity(wrist3Motor, 0.0);
    }
  }

  wb_robot_cleanup();

  return 0;
}
```

Penjelasan :

1. **#include <webots/robot.h>**: Memasukkan pustaka "robot.h" yang menyediakan fungsi-fungsi untuk menginisialisasi dan membersihkan lingkungan simulasi robot Webots.
2. **#include <webots/keyboard.h>**: Memasukkan pustaka "keyboard.h" yang menyediakan fungsi-fungsi untuk mengakses input keyboard.
3. **#include <webots/motor.h>**: Memasukkan pustaka "motor.h" yang menyediakan fungsi-fungsi untuk mengontrol motor atau sendi pada robot Webots.
4. **#include <math.h>**: Memasukkan pustaka "math.h" yang menyediakan fungsi-fungsi matematika seperti fungsi trigonometri.
5. **#define TIME_STEP 64**: Mendefinisikan konstanta **TIME_STEP** dengan nilai 64, yang merupakan interval waktu dalam milidetik antara setiap langkah simulasi.
6. **#define MAX_VELOCITY 1.0**: Mendefinisikan konstanta **MAX_VELOCITY** dengan nilai 1.0, yang merupakan kecepatan maksimum yang diizinkan untuk pergerakan sendi.
7. **#define MAX_POSITION_CHANGE 0.1**: Mendefinisikan konstanta **MAX_POSITION_CHANGE** dengan nilai 0.1, yang merupakan perubahan posisi maksimum yang diizinkan untuk setiap langkah.
8. **double clamp(double value, double min, double max)**: Mendefinisikan fungsi **clamp()** yang mengambil tiga argumen: **value** (nilai yang akan dibatasi), **min** (nilai minimum yang diizinkan), dan **max** (nilai maksimum yang diizinkan). Fungsi ini digunakan untuk membatasi suatu nilai dalam rentang tertentu dan mengembalikan nilai yang telah dibatasi.
9. **int main(int argc, char **argv)**: Memulai fungsi **main()** sebagai titik masuk utama program. Fungsi ini mengambil argumen **argc** dan **argv**, meskipun tidak digunakan dalam kode ini.
10. **wb_robot_init()**: Menginisialisasi lingkungan simulasi Webots.

11. **WbDeviceTag shoulderPanMotor = wb_robot_get_device("shoulder_pan_joint");**: Mendapatkan tag perangkat (device) motor untuk sendi "shoulder_pan_joint" pada robot. Perangkat ini akan digunakan untuk mengontrol pergerakan sendi tersebut.
12. **WbDeviceTag shoulderLiftMotor = wb_robot_get_device("shoulder_lift_joint");**: Mendapatkan tag perangkat motor untuk sendi "shoulder_lift_joint" pada robot.
13. **WbDeviceTag elbowLiftMotor = wb_robot_get_device("elbow_joint");**: Mendapatkan tag perangkat motor untuk sendi "elbow_joint" pada robot.
14. **WbDeviceTag wrist1Motor = wb_robot_get_device("wrist_1_joint");**: Mendapatkan tag perangkat motor untuk sendi "wrist_1_joint" pada robot.
15. **WbDeviceTag wrist2Motor = wb_robot_get_device("wrist_2_joint");**: Mendapatkan tag perangkat motor untuk sendi "wrist_2_joint" pada robot.
16. **WbDeviceTag wrist3Motor = wb_robot_get_device("wrist_3_joint");**: Mendapatkan tag perangkat motor untuk sendi "wrist_3_joint" pada robot.
17. **wb_keyboard_enable(TIME_STEP)**: Mengaktifkan input keyboard dengan interval **TIME_STEP** antara pembacaan input.
18. **while (wb_robot_step(TIME_STEP) != -1) { ... }**: Memulai loop while yang berjalan selama simulasi berjalan. Setiap iterasi dalam loop ini mewakili langkah simulasi sebesar **TIME_STEP** milidetik. Loop akan berhenti jika **wb_robot_step()** mengembalikan nilai -1, menunjukkan bahwa simulasi telah berakhir.
19. **int key = wb_keyboard_get_key();**: Membaca input dari keyboard dan menyimpannya dalam variabel **key** yang menunjukkan tombol apa yang ditekan.
20. Sekumpulan pernyataan **if-else** digunakan untuk mengatur pergerakan sendi berdasarkan input keyboard yang diterima. Setiap pernyataan **if** memeriksa tombol yang ditekan menggunakan variabel **key** dan melakukan perubahan posisi sendi yang sesuai menggunakan fungsi-fungsi Webots seperti **wb_motor_get_target_position()** dan **wb_motor_set_position()**. Posisi baru sendi dijaga agar tetap berada dalam rentang yang diizinkan menggunakan fungsi **clamp()**.
21. **wb_robot_cleanup()**: Membersihkan lingkungan simulasi Webots setelah simulasi selesai.
22. **return 0;**: Mengembalikan nilai 0 sebagai kode status program yang menandakan bahwa program berakhir tanpa kesalahan.