1 Vorbereitung

1.1 benötigte Bibliotheken

Universal Robots stellt mehrere Bibliotheken zur verfügung.

- libcollision.a
- libconfiguration.a
- libdev.a
- libkinematics.a
- libmath.a
- librobotcomm.a

Die Lib “librobotcomm.a” bietet die Funktionen, die nötig sind um mit dem Controller zu kommunizieren. Die dazugehörige Header Datei ist “robotinterface.h”.

1.1.1 Funktionen der Lib librobotcomm.a

inputencodinginputencoding

```
3  /* robot modes */
4  #define ROBOT_RUNNING_MODE 0
5  #define ROBOT_FREEDRIVE_MODE 1
6  #define ROBOT_READY_MODE 2
7  #define ROBOT_INITIALIZING_MODE 3
8  #define ROBOT_SECURITY_STOPPED_MODE 4
9  #define ROBOT_EMERGENCY_STOPPED_MODE 5
10 #define ROBOT_FATAL_ERROR_MODE 6
11 #define ROBOT_NO_POWER_MODE 7
12 #define ROBOT_NOT_CONNECTED_MODE 8
13 #define ROBOT_SHUTDOWN_MODE 9
```

```
14  #define ROBOT_SAFEGUARD_STOP_MODE 10

16  #ifdef __cplusplus
17  extern "C" {
18  #endif

20    uint64_t robotinterface_get_step();
21    double robotinterface_get_time();
22    double robotinterface_get_time_per_step();

24    /**
25     * \name Robot commands
26     * This is where the robot command methods start.
27     * @{
28     */
29    void robotinterface_command_position_velocity_acceleration(const double *q,
30                                                               const double *qd,
31                                                               const double
32                                                           *qdd);

34    /**
35     * \name Robot commands
36     * This is where the robot command methods start.
37     * @{
38     */
39    void robotinterface_command_velocity_acceleration(const double *q,
40                                                      const double *qd,
41                                                      const double
42                                                      *qdd);
43    void robotinterface_command_velocity(const double *qd);
44    void robotinterface_command_joint_position_velocity_acceleration(int joint,
45                                                                     double q,
46                                                                     double qd,
47                                                                     double
48                                                                     qdd);

50    void robotinterface_command_velocity_security_torque_control_torque(const
          double *qd,
51                                                                        const double *
                                                                              security_torque
                                                                              ,
52                                                                        const double *
                                                                              control_torque
                                                                              ,
53                                                                        const double *
                                                                              softness
54                                                                        );

56    void robotinterface_command_empty_command();

58    void robotinterface_master_goto_bootloader_mode();
```

```
60  /* @{ */
61  /** Value must be 0 or 1, port can be 0..9 */
62  void robotinterface_command_digital_out_port(int port, int value);
63  void robotinterface_command_digital_out_bits(unsigned short bits);
64  /** Value must be 0 or 1, port can be 0..9 */
65  void robotinterface_command_analog_out_port(int port, double value);

67  /**
68   * Value must be 0 or 1, port can be 0..9
69   * For port 0 and 1, the controller box analog inputs;
70   * - Range must be 0,1,2,3
71   * For port 2 and 3, the tool board analog inputs;
72   * - Range 0 is 0..5 Volt
73   * - Range 1 is 0..10 Volts
74   * - Range 4 is 4..20 mA
75   */
76  void robotinterface_command_analog_input_range_port(int port, int range);
77  void robotinterface_command_analog_output_domain(int port, int type);
78  void robotinterface_command_tool_output_voltage(unsigned char value);

80  /* The TCP is regarded as part of the robot
81   * Get and set methods for TCP and TCP payload are as follows
82   */
83  void robotinterface_set_tcp(const double *tcp_pose);
84  void robotinterface_set_tcp_payload(double tcp_payload);
85  void robotinterface_get_tcp(double *tcp_pose);
86  double robotinterface_get_tcp_payload();
87  /* @} */
88  /* @} */

90  /* Funtions to set/get force and torque (wrench) at the TCP */
91  void robotinterface_set_tcp_wrench(const double *new_tcp_wrench, const int
        in_base_coord);
92  void robotinterface_get_tcp_wrench(double *gotten_tcp_wrench);
93  int robotinterface_is_tcp_wrench_in_base_coord();



96  /**
97   * \name Communication
98   * This is where the robot communcation methods start.
99   *
100  * \{
101  * \brief Initialize connection to the robot.
102  *
103  * If robot is directly connected, it returns TRUE,
104  * otherwise it will return false and enter simulation mode.
105  *
106  * simulation mode can also be simulated by setting argument
107  * simulated = 1.
108  */
109 int robotinterface_open(int open_simulated);
110 int robotinterface_do_open(int open_simulated);
```

```
111 int robotinterface_open_allow_bootloader(int open_simulated);
112 void robotinterface_read_state_blocking();
113 void robotinterface_send();

115 /* ! physical robot connected? (Ethernet talking to coldfire) */
116 int robotinterface_is_robot_connected();


119 /* ! Shuts down the communication channel to the robot. */
120 int robotinterface_close();


123 /**
124  * Returns robot mode, which is one of the following:
125  *
126  * - ROBOT_RUNNING_MODE
127  * - ROBOT_READY_MODE
128  * - ROBOT_INITIALIZING_MODE
129  * - ROBOT_STOPPED_MODE
130  * - ROBOT_SECURITY_STOPPED_MODE
131  * - ROBOT_FATAL_ERROR_MODE
132  * - ROBOT_NOT_CONNECTED_MODE
133  */
134 uint8_t robotinterface_get_robot_mode();

136 /**
137  * Sets the robot in ROBOT_READY_MODE if current mode is ROBOT_RUNNING_MODE
138  */
139 void robotinterface_set_robot_ready_mode();

141 /**
142  * Sets the robot in ROBOT_RUNNING_MODE if current mode is
143  * ROBOT_READY_MODE or ROBOT_FREEDRIVE_MODE
144  *
145  */
146 void robotinterface_set_robot_running_mode();

148 /**
149  * Sets the robot in ROBOT_FREEDRIVE_MODE if current mode is
150  *  ROBOT_RUNNING_MODE
151  *
152  */
153 void robotinterface_set_robot_freedrive_mode();


156 /**
157  * Returns the state of one joint.
158  *
159  * Each joint can be in one of the
160  * following states.
161  *
162  * - JOINT_MOTOR_INITIALISATION_MODE
```

```
163   * - JOINT_BOOTING_MODE
164   * - JOINT_DEAD_COMMUTATION_MODE
165   * - JOINT_BOOTLOADER_MODE
166   * - JOINT_CALIBRATION_MODE
167   * - JOINT_STOPPED_MODE
168   * - JOINT_FAULT_MODE
169   * - JOINT_RUNNING_MODE
170   * - JOINT_INITIALISATION_MODE
171   * - JOINT_IDLE_MODE
172   */
173  uint8_t robotinterface_get_joint_mode(int joint);

175  /**
176   * Returns the state of the tool
177   *  Either:
178   * - JOINT_IDLE_MODE
179   * - JOINT_BOOTLOADER_MODE
180   * - JOINT_RUNNING_MODE
181   */
182  uint8_t robotinterface_get_tool_mode();

184  /**
185   * Returns number_of_message that needs to be read by
186   * #robotinterface_get_next_message().
187   */
188  uint8_t robotinterface_get_message_count();

190  /* Revisit this list when making a new logger system */
191  /* Source is:
192   69: Safety sys 2
193   68: Euromap67 2
194   67: Euromap67 1
195   66: Teach Pendant 2
196   65: Teach Pendant 1
197   6: Tool
198   5: Wrist 3
199   4: Wrist 2
200   3: Wrist 1
201   2: Elbow
202   1: Shoulder
203   0: Base
204  -1: Safetysys (errors reported by masterboard code
205  -2: Controller
206  -3: RTMachine
207  -4: Simulated Robot
208  -5: GUI
209  -6: ControlBox (not used, was used in a Version mismatch check in the
         controller, the source for that message has been replaced with -2)
210  */

212  struct message_t {
213    uint64_t timestamp;
```

```
214     char source;
215     char *text;
216   };

218   /**
219    * Returns length of message copied to the char*
220    */
221   int robotinterface_get_message(struct message_t *message);

223   /**
224    * Takes a message as error codes rathern than text
225    */
226   int robotinterface_get_message_codes(struct message_t *msg, int *error_code,
227                                        int *error_argument);

229   /*@}*/

231   /**
232    * \name Security stops
233    *
234    * This is where the robot security stop methods start.
235    * <em>Implemented in sequritycheck.c</em>
236    *@{
237    */

239   /**
240    * \return 0 if there is no error in the robot, true otherwise.
241    */
242   void robotinterface_power_on_robot();
243   void robotinterface_power_off_robot();
244   void robotinterface_security_stop(char joint_code, int error_state,
245                                     int error_argument);
246   int robotinterface_is_power_on_robot();
247   int robotinterface_is_security_stopped();
248   int robotinterface_is_emergency_stopped();
249   int robotinterface_is_extra_button_pressed(); /* The button on the back side
          of the screen */
250   int robotinterface_is_power_button_pressed();  /* The big green button on the
           controller box */
251   int robotinterface_is_safety_signal_such_that_we_should_stop(); /* This is
          from the safety stop interface */
252   int robotinterface_unlock_security_stop();
253   int robotinterface_has_security_message();
254   int robotinterface_get_security_message(struct message_t *message,
255                                           int *error_state,
256                                           int *error_argument);
257   uint32_t robotinterface_get_master_control_bits();

259   /*@}*/

261   /**
262    * \name Robot methods
```

```
263     *
264     * This is where actual robot methods start.
265     *
266     * Each function copies the resulting joint values of the
267     * robot to the supplied array.
268     *@{
269     */
270    void robotinterface_get_actual(double *q, double *qd);
271    void robotinterface_get_actual_position(double *q);
272    void robotinterface_get_actual_velocity(double *qd);
273    void robotinterface_get_actual_current(double *I);
274    void robotinterface_get_actual_accelerometers(double *ax, double *ay,
275                                                  double *az);
276    double robotinterface_get_tcp_force_scalar();
277    void robotinterface_get_tcp_force(double *F);
278    void robotinterface_get_tcp_speed(double *V);
279    double robotinterface_get_tcp_power();
280    double robotinterface_get_power();

282    /*@}*/

284    int robotinterface_get_actual_digital_input(int port);
285    int robotinterface_get_actual_digital_output(int port);
286    unsigned short robotinterface_get_actual_digital_input_bits();
287    unsigned short robotinterface_get_actual_digital_output_bits();
288    double robotinterface_get_actual_analog_input(int port);
289    double robotinterface_get_actual_analog_output(int port);
290    unsigned char robotinterface_get_actual_analog_input_range(int port);
291    unsigned char robotinterface_get_actual_analog_output_domain(int port);

293    /** \name Ideal methods
294     * This is where target (ideal) methods start.
295     */
296    /*@{*/
297    void robotinterface_get_target(double *q, double *qd, double *qdd);
298    void robotinterface_get_target_position(double *q);
299    void robotinterface_get_target_velocity(double *qd);
300    void robotinterface_get_target_acceleration(double *qdd);
301    void robotinterface_get_target_current(double *I);
302    void robotinterface_get_target_moment(double *m);

304    unsigned short robotinterface_get_target_digital_output_bits();
305    double robotinterface_get_target_analog_output(int port);
306    unsigned char robotinterface_get_target_analog_input_range(int port);
307    unsigned char robotinterface_get_target_analog_output_domain(int port);

309    /*@}*/



312    /**
313     *
314     * \name Temperature, Voltage and Current Methods
```

```
315   */
316   /*@{*/

318   /* master */
319   float robotinterface_get_master_temperature();  /* new */
320   float robotinterface_get_robot_voltage_48V(); /* new */
321   float robotinterface_get_robot_current(); /* new */
322   float robotinterface_get_master_io_current(); /* new */
323   unsigned char robotinterface_get_master_safety_state();
324   unsigned char robotinterface_get_master_on_off_state();
325   void robotinterface_safely_remove_euromap67_hardware();
326   void robotinterface_disable_teach_pendant_safety();

328   /* joint */
329   void robotinterface_get_motor_temperature(float *T);
330   void robotinterface_get_micro_temperature(float *T);
331   void robotinterface_get_joint_voltage(float *V);

333   /* tool */
334   float robotinterface_get_tool_temperature();
335   float robotinterface_get_tool_voltage_48V();
336   unsigned char robotinterface_get_tool_output_voltage();
337   float robotinterface_get_tool_current();

339   /* Euromap for CB2.0*/
340   uint8_t robotinterface_is_euromap_hardware_installed();
341   uint8_t robotinterface_get_euromap_input(uint8_t input_bit_number);
342   uint8_t robotinterface_get_euromap_output(uint8_t output_bit_number);
343   void robotinterface_set_euromap_output(uint8_t output_bit_number, uint8_t
          desired_value);
344   uint32_t robotinterface_get_euromap_input_bits();
345   uint32_t robotinterface_get_euromap_output_bits();
346   uint16_t robotinterface_get_euromap_24V_voltage();
347   uint16_t robotinterface_get_euromap_24V_current();

349   /* General purpose registers */
350   uint16_t robotinterface_get_general_purpose_register(int address);
351   void robotinterface_set_general_purpose_register(int address, uint16_t value)
          ;

353   /*@}*/
```

1.2 Compilieren

Universal Robots bietet eine Beispielapplikation an. Diese Wird mit Scons Compiliert. Scons ist ein Python Programm Folgend der Code zum Compilieren der main Dabei und das Linken dre Libs

```
inputencodinginputencoding
```

```
# Import construction environment
Import('env')
# cppath and libpath has to be adjusted to match the library directory of
    librobotcomm
cpppath = ['../libs']
libpath = ['../libs']
libs = ['robotcomm', 'kinematics', 'configuration', 'dev', 'collision', 'm', '
    math', 'pthread']
ccflags = ['-O2', '-Wall', '-g']
env = Environment(CC = 'g++',
      CPPPATH = cpppath,
      LIBPATH = libpath,
      LIBS = libs,
      CCFLAGS = ccflags,
      CPPDEFINES = 'NDEBUG')

base_src = ['../libs/base_utils.c', '../libs/startup_utils.c', '../libs/
    interrupt_utils.c', '../libs/ur5lib.c', '../libs/helper.c', '../libs/
    tcphelper.c', '../libs/ur_kin.c']
tcp_src = ['main.c'] + base_src

tcp_prog = env.Program(target='ur5-server', source=tcp_src)

env.Default(tcp_prog)

################################################################################
```

2 Berechnung einer PTP Bahn

Die Formeln für die Berechnung sind aus der Lektüre “Industrieroboter: Methoden der Steuerung und Regelung ” von Wolfgang Weber 2. Auflage, herrausgebracht von Carl Hanser Verlag GmbH & Co. KG.

Um die Ergebnisse der Berechnungen zu benutzen, werden die ergebnisse in ein selbst geschriebenes Struct zugewiesen.

inputencodinginputencoding

```
// Momentan ist nicht die Orientierung des mit einbezogen!!

typedef struct MoveLinearPacket{
    PVAPacket pva; // struct das für alle Gelenke die position, Geschwindigkeit und Beschleunigung enthällt
    double a_max; // maximale beschleunigung des leitjoints
    double v_max; // maximale geschwindigkeit des leitjoints
    double te; // Zielzeitpunkt
    double tv; // bremsphase start
    double tb;  // beschleunigungsphase
    double qst[3]; // Start Koordinaten im raum(kartesisch) des TCP
    double qz[3]; // ziel Koordinaten im raum(kartesisch)
    double T[16]; // beinhält die 16 möglichen forwärtstransformationen nach der interpolation
    double se; // streckenlänge
    double acceleration; // beschleunigung des TCP
    double velocity; // Geschwindigkeit des TCP
    double position;  // Aktuelle position des TCP auf der Strecke Nicht Kartesisch
    double point_in_time;// wärend der interpolation enthällt diese variable die aktuelle interpolationszeit
    int interpolations;  // anzahl an interpolationen
} MoveLinearPacket;

typedef struct MovePTPPacket{
    PVAPacket pva; // struct das für alle Gelenke die position, Geschwindigkeit und Beschleunigung enthällt
    JointVector a_max; // maximale beschleunigung für die 6 joints
    JointVector v_max; // maximale geschwindigkeiten für die 6 joints
    JointVector te; // Zeitpunkt am ende der interpolation
    JointVector tb; // Zeitpunkt für das ende der Beschleunigungsphasen der 6 joints
```

```
28      JointVector tv; // Zeitpunkt für den anfang der Bremsphasen der 6 joints
29      JointVector signse; // richtung der joints
30      JointVector qst; // Startpositionen der 6.Joints
31      JointVector se; // streckenlänge der einzelnen joints
32      JointVector qz; // zielpositionen der 6. Joints
33      double point_in_time; // wärend der interpolation enthällt diese variable
            die aktuelle interpolationszeit
34      int interpolations; // anzahl an interpolationen
35  } MovePTPPacket;
```

inputencodinginputencoding

```
2   // q is actual position of the six joints
3   // qz is the target position of the six joints
4   // move_pva struct will contain the final calculations for the
        interpolationphase
5   // profil  determines if the acceleration profil is a sinoidenprofil or eiter a
         rampenprofil

7   void calc_ptp_profile(const double * q, double * pz, MovePTPPacket *move_pva,
        int profil){
8       //    double T[16];
9       double factor= (profil == RAMPENPROFIL) ?  1.0 : 2.0;
10      if(debug)
11          printf("factor %f, profil %s\n", factor, profil == RAMPENPROFIL ? "
                Rampenprofil" : "Sinoidenprofil");
12      double se= 0.0;
13      int i, l=0;

15      for(i=0; i < 6; i++){

17  // Berechnung der Benötigten Strecke anhand der vorgegebenen Geschwindigkeits
        und Beschleunigungswerte für jeden einzelnen Joint und ausrechnung des
        Leitjoints um eine Synchrone Bahn zu berechnen.

19          move_pva->qst[i]=q[i];
20          move_pva->qz[i] =pz[i];
21          move_pva->v_max[i]= deg_to_rad(VELOCITY_RAD_MAX);
22          move_pva->a_max[i]= deg_to_rad(ACCELERATION_RAD_MAX);
23          se = pz[i] - q[i];
24          move_pva->signse[i]= se < 0.0 ? -1.0 : (se > 0  ? 1.0 : 0.0);
25          move_pva->se[i] = fabs(se);

27  // Falls die Strecke so minimal ist, das davon ausgegangen wird,
28  // dass der joint schon an der richtigen Position steht, wird für diesen Joint
        keine Bahn berechnet
29          if(move_pva->se[i] < deg_to_rad(EPS)){
30              move_pva->tb[i]=0.0;
31              move_pva->te[i]=0.0;
32              move_pva->tv[i]=0.0;
33              move_pva->v_max[i]=0.0;
34              move_pva->a_max[i]=0.0;
35          }else{
```

```
37 // anpassung der Geschwindigkeit, falls die Strecke zu kurz ist um auf die
       eigentliche Geschwindigkeit zu kommen.
38             if(move_pva->v_max[i] > sqrt((move_pva->se[i]*move_pva->a_max[i])/
                   factor))
39                 move_pva->v_max[i] = sqrt((move_pva->se[i]*move_pva->a_max[i])/
                       factor);

41 // Berechnung anhand der Beschleunigungs und bremsphasen der einzelnen joints
42             move_pva->tb[i] = (factor*move_pva->v_max[i])/move_pva->a_max[i];
43             move_pva->te[i] = (move_pva->se[i] / move_pva->v_max[i]) + move_pva
                   ->tb[i];
44             move_pva->tv[i] = move_pva->te[i] - move_pva->tb[i];
45         }

47 // l ist der leitjoint, derjenige der am meisten zeit für seine strecke benö
       tigt
48         if(move_pva->te[i] > move_pva->te[l]){
49             l=i;
50         }
51     }
52 // Falls Joint berechnet wird, anpassen der einznene Phasen an den
       Iterpolationstakt
53     if(move_pva->se[l] < deg_to_rad(EPS)){
54         // adapt longest joint to interpolationphase
55         move_pva->tb[l] = (floor((factor*move_pva->v_max[l])/(move_pva->a_max[l
               ]*T_IPO))+1)*T_IPO;
56         move_pva->tv[l] = (floor(move_pva->se[l]/(move_pva->v_max[l]*T_IPO))+1)
               *T_IPO;
57         move_pva->te[l] = move_pva->tv[l]+move_pva->tb[l];
58         move_pva->v_max[l] = move_pva->se[l]/move_pva->tv[l];
59         move_pva->a_max[l] = (factor*move_pva->se[l])/(move_pva->tv[l]*move_pva
               ->tb[l]);
60     }

62 //     adapt te of all joints to te of longest joint
63     for(i=0;i<6;i++){
64         if(l != i && move_pva->se[i] > deg_to_rad(EPS)){
65             move_pva->te[i] = move_pva->te[l];
66             if(profil == RAMPENPROFIL){
67                 move_pva->v_max[i] = (move_pva->a_max[i]*move_pva->te[l])/2 -
                       sqrt(((POW(move_pva->a_max[i])*POW(move_pva->te[l]))/4)-(
                       move_pva->se[i]*move_pva->a_max[i]));
68             }else{
69                 move_pva->v_max[i] = (move_pva->a_max[i]*move_pva->te[l])/4 -
                       sqrt(((POW(move_pva->a_max[i])*POW(move_pva->te[l]))-(8*
                       move_pva->se[i]*move_pva->a_max[i]))/16);
70             }

72             if(move_pva->v_max[i] > sqrt((move_pva->se[i]*move_pva->a_max[i])/
                   factor))
73                 move_pva->v_max[i] = sqrt((move_pva->se[i]*move_pva->a_max[i])/
```

```
                    factor);

75              move_pva->tv[i] = floor(move_pva->se[i]/(move_pva->v_max[i]*T_IPO))
                    *T_IPO;
76              move_pva->tb[i] = move_pva->te[i] - move_pva->tv[i];
77              move_pva->v_max[i] = move_pva->se[i] / move_pva->tv[i];
78              move_pva->a_max[i] = (factor*move_pva->v_max[i]) /  move_pva->tb[i
                    ];
79          }
80      }

82      // Wenn mind. Ein joint über die schwelle der 0.2grad hinausgeht, werden
            die maximalen intrepolationen berechnet, ansohnsten gibt es 0
            Interpolationen, der Roboter steht schon an der Zielstelle.
83      move_pva->interpolations= move_pva->se[l] < deg_to_rad(EPS) ? 0 :
84                                                                  (int) round(
                                                                        move_pva->
                                                                        te[l] /
                                                                        T_IPO);



87      if(debug) print_ptp_debug(move_pva, profil);

89      return;
90  }
```

3 Beispielprogramme

3.1 Interpolieren und PTP Bahn abfahren mit der C-API

inputencodinginputencoding

```
1     PVAPacket pva;
2     MovePTPPacket move_pva_packet;
3     bzero(&move_pva_packet,sizeof(move_pva_packet));
4     int profil = RAMPENPROFIL;

6     // get the actual position so we can be sure where to start calculating the
          ptp profile
7     robotinterface_read_state_blocking();
8     robotinterface_get_actual_position(pva.position);
9     memcpy(&last_pva_packet, &pva_packet, sizeof(pva_packet));
10    robotinterface_command_velocity(zero_vector);
11    robotinterface_send();

13    // calculate ptp profile
14    calc_ptp_profile(pva.position, qz, &move_pva_packet, profil);

16    int i;

18    for(i=0; i < move_pva_packet.interpolations+1; i++){
19        robotinterface_read_state_blocking();

21        // falls ein Sicherheitsstopp erfolgt, bewegung abbrechen
22        if(robotinterface_is_security_stopped()) {
23            robotinterface_get_actual_current(currents_actual);
24            robotinterface_get_target_current(currents_target);
25            printf("security stopped at interpolation: %d\n", i);
26            robotinterface_command_empty_command();
27            robotinterface_send();
28            break;
29        }
30        //  Zeitpunkt in der Interpolation berechnen
31        move_pva_packet.point_in_time= (double) i * T_IPO;

33        // Interpoliere mit Sinoidenprofil
34        interpolation_sin_ptp(&move_pva_packet);
```

```
36          // Werte der Position, Geschwindigkeit und Beschleunigung des aktuellen
                 Interpolationsschritts an Roboter übergeben
37          robotinterface_command_position_velocity_acceleration(move_pva_packet.
                pva.position,
38                                                                  move_pva_packet.
                                                                        pva.velocity,
39                                                                  move_pva_packet.
                                                                        pva.
                                                                        acceleration)
                                                                        ;
40          // befehle an roboter senden
41          robotinterface_send();
42      }
43      // Zur Sicherheit nochmal Roboter zum Stillstand bringen
44      for(i=0;i<10;i++){
45          robotinterface_read_state_blocking();
46          robotinterface_command_position_velocity_acceleration(pva.position,
                zero_vector, zero_vector);
47          robotinterface_send();
48      }
```