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

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

```
#define ROBOT_SAFEGUARD_STOP_MODE 10

#ifdef __cplusplus
extern "C" {
#endif

  uint64_t robotinterface_get_step();
  double robotinterface_get_time();
  double robotinterface_get_time_per_step();

  /**
   * \name Robot commands
   * This is where the robot command methods start.
   * @{
   */
  void robotinterface_command_position_velocity_acceleration(const double *q,
                                                             const double *qd,
                                                             const double
                                                       *qdd);

  /**
   * \name Robot commands
   * This is where the robot command methods start.
   * @{
   */
  void robotinterface_command_velocity_acceleration(const double *q,
                                                             const double *qd,
                                                             const double
                                                             *qdd);
  void robotinterface_command_velocity(const double *qd);
  void robotinterface_command_joint_position_velocity_acceleration(int joint,
                                                                   double q,
                                                                   double qd,
                                                                   double
                                                                   qdd);

  void robotinterface_command_velocity_security_torque_control_torque(const
      double *qd,
                                                             const double *
                                                                 security_torque
                                                                 ,
                                                             const double *
                                                                 control_torque
                                                                 ,
                                                             const double *
                                                                 softness
                                                             );

  void robotinterface_command_empty_command();

  void robotinterface_master_goto_bootloader_mode();
```

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

/**
 * Value must be 0 or 1, port can be 0..9
 * For port 0 and 1, the controller box analog inputs;
 * - Range must be 0,1,2,3
 * For port 2 and 3, the tool board analog inputs;
 * - Range 0 is 0..5 Volt
 * - Range 1 is 0..10 Volts
 * - Range 4 is 4..20 mA
 */
void robotinterface_command_analog_input_range_port(int port, int range);
void robotinterface_command_analog_output_domain(int port, int type);
void robotinterface_command_tool_output_voltage(unsigned char value);

/* The TCP is regarded as part of the robot
 * Get and set methods for TCP and TCP payload are as follows
 */
void robotinterface_set_tcp(const double *tcp_pose);
void robotinterface_set_tcp_payload(double tcp_payload);
void robotinterface_get_tcp(double *tcp_pose);
double robotinterface_get_tcp_payload();
/* @} */
/* @} */

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


/**
 * \name Communication
 * This is where the robot communcation methods start.
 *
 * \{
 * \brief Initialize connection to the robot.
 *
 * If robot is directly connected, it returns TRUE,
 * otherwise it will return false and enter simulation mode.
 *
 * simulation mode can also be simulated by setting argument
 * simulated = 1.
 */
int robotinterface_open(int open_simulated);
int robotinterface_do_open(int open_simulated);
```

```
int robotinterface_open_allow_bootloader(int open_simulated);
void robotinterface_read_state_blocking();
void robotinterface_send();

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


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


/**
 * Returns robot mode, which is one of the following:
 *
 * - ROBOT_RUNNING_MODE
 * - ROBOT_READY_MODE
 * - ROBOT_INITIALIZING_MODE
 * - ROBOT_STOPPED_MODE
 * - ROBOT_SECURITY_STOPPED_MODE
 * - ROBOT_FATAL_ERROR_MODE
 * - ROBOT_NOT_CONNECTED_MODE
 */
uint8_t robotinterface_get_robot_mode();

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

/**
 * Sets the robot in ROBOT_RUNNING_MODE if current mode is
 * ROBOT_READY_MODE or ROBOT_FREEDRIVE_MODE
 *
 */
void robotinterface_set_robot_running_mode();

/**
 * Sets the robot in ROBOT_FREEDRIVE_MODE if current mode is
 *  ROBOT_RUNNING_MODE
 *
 */
void robotinterface_set_robot_freedrive_mode();


/**
 * Returns the state of one joint.
 *
 * Each joint can be in one of the
 * following states.
 *
 * - JOINT_MOTOR_INITIALISATION_MODE
```

```
 * - JOINT_BOOTING_MODE
 * - JOINT_DEAD_COMMUTATION_MODE
 * - JOINT_BOOTLOADER_MODE
 * - JOINT_CALIBRATION_MODE
 * - JOINT_STOPPED_MODE
 * - JOINT_FAULT_MODE
 * - JOINT_RUNNING_MODE
 * - JOINT_INITIALISATION_MODE
 * - JOINT_IDLE_MODE
 */
uint8_t robotinterface_get_joint_mode(int joint);

/**
 * Returns the state of the tool
 *  Either:
 * - JOINT_IDLE_MODE
 * - JOINT_BOOTLOADER_MODE
 * - JOINT_RUNNING_MODE
 */
uint8_t robotinterface_get_tool_mode();

/**
 * Returns number_of_message that needs to be read by
 * #robotinterface_get_next_message().
 */
uint8_t robotinterface_get_message_count();

/* Revisit this list when making a new logger system */
/* Source is:
 69: Safety sys 2
 68: Euromap67 2
 67: Euromap67 1
 66: Teach Pendant 2
 65: Teach Pendant 1
 6: Tool
 5: Wrist 3
 4: Wrist 2
 3: Wrist 1
 2: Elbow
 1: Shoulder
 0: Base
-1: Safetysys (errors reported by masterboard code
-2: Controller
-3: RTMachine
-4: Simulated Robot
-5: GUI
-6: ControlBox (not used, was used in a Version mismatch check in the
    controller, the source for that message has been replaced with -2)
*/

struct message_t {
  uint64_t timestamp;
```

```
    char source;
    char *text;
  };

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

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

  /*@}*/

  /**
   * \name Security stops
   *
   * This is where the robot security stop methods start.
   * <em>Implemented in sequritycheck.c</em>
   *@{
   */

  /**
   * \return 0 if there is no error in the robot, true otherwise.
   */
  void robotinterface_power_on_robot();
  void robotinterface_power_off_robot();
  void robotinterface_security_stop(char joint_code, int error_state,
                                    int error_argument);
  int robotinterface_is_power_on_robot();
  int robotinterface_is_security_stopped();
  int robotinterface_is_emergency_stopped();
  int robotinterface_is_extra_button_pressed(); /* The button on the back side of the screen */
  int robotinterface_is_power_button_pressed();  /* The big green button on the controller box */
  int robotinterface_is_safety_signal_such_that_we_should_stop(); /* This is from the safety stop interface */
  int robotinterface_unlock_security_stop();
  int robotinterface_has_security_message();
  int robotinterface_get_security_message(struct message_t *message,
                                          int *error_state,
                                          int *error_argument);
  uint32_t robotinterface_get_master_control_bits();

  /*@}*/

  /**
   * \name Robot methods
```

```
 *
 * This is where actual robot methods start.
 *
 * Each function copies the resulting joint values of the
 * robot to the supplied array.
 *@{
 */
void robotinterface_get_actual(double *q, double *qd);
void robotinterface_get_actual_position(double *q);
void robotinterface_get_actual_velocity(double *qd);
void robotinterface_get_actual_current(double *I);
void robotinterface_get_actual_accelerometers(double *ax, double *ay,
                                              double *az);
double robotinterface_get_tcp_force_scalar();
void robotinterface_get_tcp_force(double *F);
void robotinterface_get_tcp_speed(double *V);
double robotinterface_get_tcp_power();
double robotinterface_get_power();

/*@}*/

int robotinterface_get_actual_digital_input(int port);
int robotinterface_get_actual_digital_output(int port);
unsigned short robotinterface_get_actual_digital_input_bits();
unsigned short robotinterface_get_actual_digital_output_bits();
double robotinterface_get_actual_analog_input(int port);
double robotinterface_get_actual_analog_output(int port);
unsigned char robotinterface_get_actual_analog_input_range(int port);
unsigned char robotinterface_get_actual_analog_output_domain(int port);

/** \name Ideal methods
 * This is where target (ideal) methods start.
 */
/*@{*/
void robotinterface_get_target(double *q, double *qd, double *qdd);
void robotinterface_get_target_position(double *q);
void robotinterface_get_target_velocity(double *qd);
void robotinterface_get_target_acceleration(double *qdd);
void robotinterface_get_target_current(double *I);
void robotinterface_get_target_moment(double *m);

unsigned short robotinterface_get_target_digital_output_bits();
double robotinterface_get_target_analog_output(int port);
unsigned char robotinterface_get_target_analog_input_range(int port);
unsigned char robotinterface_get_target_analog_output_domain(int port);

/*@}*/


/**
 *
 * \name Temperature, Voltage and Current Methods
```

```
 */
/*@{*/

/* master */
float robotinterface_get_master_temperature();  /* new */
float robotinterface_get_robot_voltage_48V(); /* new */
float robotinterface_get_robot_current(); /* new */
float robotinterface_get_master_io_current(); /* new */
unsigned char robotinterface_get_master_safety_state();
unsigned char robotinterface_get_master_on_off_state();
void robotinterface_safely_remove_euromap67_hardware();
void robotinterface_disable_teach_pendant_safety();

/* joint */
void robotinterface_get_motor_temperature(float *T);
void robotinterface_get_micro_temperature(float *T);
void robotinterface_get_joint_voltage(float *V);

/* tool */
float robotinterface_get_tool_temperature();
float robotinterface_get_tool_voltage_48V();
unsigned char robotinterface_get_tool_output_voltage();
float robotinterface_get_tool_current();

/* Euromap for CB2.0*/
uint8_t robotinterface_is_euromap_hardware_installed();
uint8_t robotinterface_get_euromap_input(uint8_t input_bit_number);
uint8_t robotinterface_get_euromap_output(uint8_t output_bit_number);
void robotinterface_set_euromap_output(uint8_t output_bit_number, uint8_t
    desired_value);
uint32_t robotinterface_get_euromap_input_bits();
uint32_t robotinterface_get_euromap_output_bits();
uint16_t robotinterface_get_euromap_24V_voltage();
uint16_t robotinterface_get_euromap_24V_current();

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

/*@}*/
```

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
```

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

inputencodinginputencoding

```

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

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

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

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

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

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

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

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

// l ist der leitjoint, derjenige der am meisten zeit für seine strecke benötigt
        if(move_pva->te[i] > move_pva->te[l]){
            l=i;
        }
    }
// Falls Joint berechnet wird, anpassen der einznene Phasen an den Iterpolationstakt
    if(move_pva->se[l] < deg_to_rad(EPS)){
        // adapt longest joint to interpolationphase
        move_pva->tb[l] = (floor((factor*move_pva->v_max[l])/(move_pva->a_max[l]*T_IPO))+1)*T_IPO;
        move_pva->tv[l] = (floor(move_pva->se[l]/(move_pva->v_max[l]*T_IPO))+1)*T_IPO;
        move_pva->te[l] = move_pva->tv[l]+move_pva->tb[l];
        move_pva->v_max[l] = move_pva->se[l]/move_pva->tv[l];
        move_pva->a_max[l] = (factor*move_pva->se[l])/(move_pva->tv[l]*move_pva->tb[l]);
    }

//     adapt te of all joints to te of longest joint
    for(i=0;i<6;i++){
        if(l != i && move_pva->se[i] > deg_to_rad(EPS)){
            move_pva->te[i] = move_pva->te[l];
            if(profil == RAMPENPROFIL){
                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]));
            }else{
                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);
            }

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

```
                    factor);

            move_pva->tv[i] = floor(move_pva->se[i]/(move_pva->v_max[i]*T_IPO))*T_IPO;
            move_pva->tb[i] = move_pva->te[i] - move_pva->tv[i];
            move_pva->v_max[i] = move_pva->se[i] / move_pva->tv[i];
            move_pva->a_max[i] = (factor*move_pva->v_max[i]) /  move_pva->tb[i];
        }
    }

    // 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.
    move_pva->interpolations= move_pva->se[l] < deg_to_rad(EPS) ? 0 :
                                                              (int) round(move_pva->te[l] / T_IPO);



    if(debug) print_ptp_debug(move_pva, profil);

    return;
}
```

3 Beispielprogramme

3.1 Interpolieren und PTP Bahn abfahren mit der C-API

inputencodinginputencoding

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

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

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

    int i;

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

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

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

```
        // Werte der Position, Geschwindigkeit und Beschleunigung des aktuellen Interpolationsschritts an Roboter übergeben
        robotinterface_command_position_velocity_acceleration(move_pva_packet.pva.position,
                                                              move_pva_packet.pva.velocity,
                                                              move_pva_packet.pva.acceleration);
        // befehle an roboter senden
        robotinterface_send();
    }
    // Zur Sicherheit nochmal Roboter zum Stillstand bringen
    for(i=0;i<10;i++){
        robotinterface_read_state_blocking();
        robotinterface_command_position_velocity_acceleration(pva.position, zero_vector, zero_vector);
        robotinterface_send();
    }
```