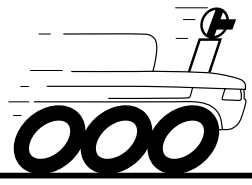
Pan-Tilt



USER MANUAL

Version 2.0

K-Team S.A.

Lausanne, 20 February 2001

Contents

1. GETTING STARTED 2

2. USING THE CAMERA STANDALONE 3

2.1. USING THE FUNCTIONS FROM A TERMINAL 3
2.2. WRITING A C PROGRAM 3

3. USING A KOALA 4

3.1. USING A TERMINAL 5
3.2. WRITING A C PROGRAM 5

4. PAN-TILT COMMAND PROTOCOL 6

5. SOFTWARE EXAMPLES 7

1. Getting Started

Pan-tilt camera systems are shipped with a Kameleon SBC controller and a custom REB (Robotics Extension Board) interface. The camera(s) can be used stand-alone in any custom application, or can be interfaced with a Koala mobile robot as shown in Figure 1a.

If the REB is not already connected to the Kameleon, install it by lining up the two double rows of pins under the REB with the connection holes on the Kameleon. Squeeze the REB and Kameleon together firmly, being careful not to bend any of the pins.

The two motors from the camera plug into the provided ports on the REB, as shown in Figure 1b. If you have two pan-tilt cameras the REB will have two additional ports for motors 2 and 3 (not shown). Each camera also has a video cable and a power cable, the form of which depends on your system configuration. Please proceed to section 2 if you are using the camera in standalone mode, or to section 3 if you are connecting the camera to a Koala robot.

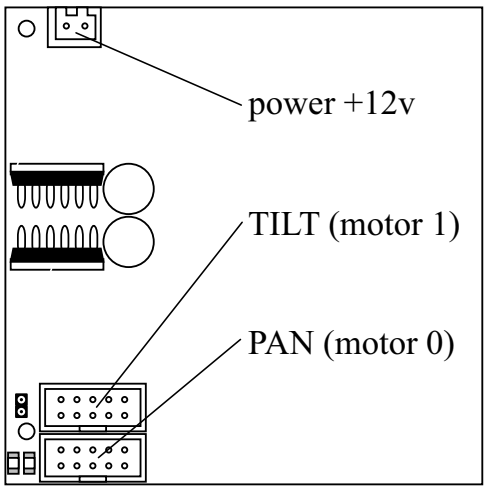


Figure 1a (left) – Double Pan-Tilt Camera mounted on Koala robot
Figure 1b (right) – REB motor ports 0 and 1

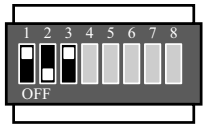
2. Using the Camera Standalone

Mount the Kameleon securely to your desired system using its four mounting holes, and then fix the camera using the M3 threaded hole on its bottom plate. Connect the camera to the REB as described in section 1. Plug the power cable from the camera into the corresponding port on the REB, and connect the RCA video output jack to your monitor or frame-grabber. Now connect the two power cables to the Kameleon (8-pin AMP connector) and to the REB (2-pin Wago connector), and hook them up to a 12v DC source.

You may either command the camera from a simple ASCII terminal via the serial port, or write your own custom C program. It is recommended you first test the camera from a terminal before writing your custom C program.

2.1. Using the Functions from a Terminal

First you must set the Kameleon into Serial Communications (SerCom) mode by adjusting the DIP switches located near the power connector as shown in figure 2.



SerCom 38400

Figure 2 – DIP switch modes

You may use any standard ASCII terminal program to communicate with the Kameleon (e.g. HyperTerminal for Windows, Microphone for Mac, Minicom for Linux). Ensure it is properly configured for 38400 baud, 8 bit, no parity, and 1 stop bit. Connect a standard serial cable to the Kameleon 9-pin DSUB serial port connector and turn on the power to the Kameleon. You should see the following on your terminal screen (the version number may vary depending on your BIOS version):

```
ROM of K376SBC, (C) E.Franzi, P.Arnaud, K-Team SA, R1.15 (11 Jul 2000)
REB support by C.Jaquet, K-Team SA
Serial Communication Protocol
```

A test program has been stored in the Kameleon's flash memory. To run it, type: **`run user-flash`**

The pan-tilt camera should perform an automatic calibration and center itself when finished. A simple ASCII protocol is provided for commanding the motors either from a terminal or from another device communicating over the serial port. Please see section 4 for a description of this protocol.

2.2. Writing a C Program

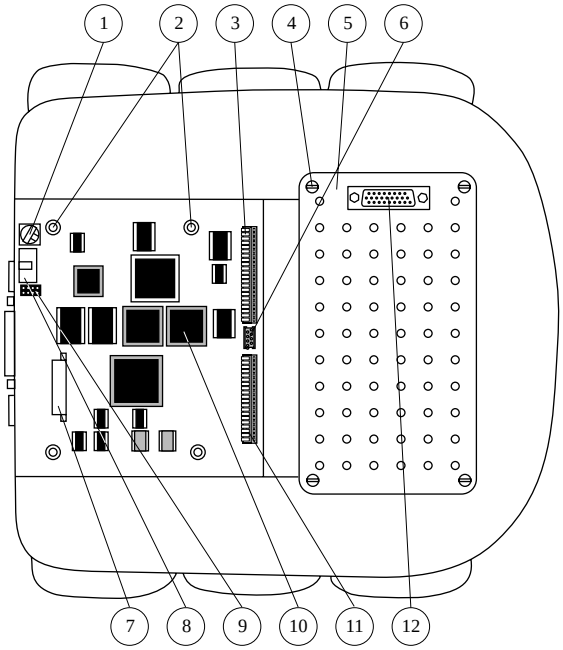
The source code of the above-mentioned test program (`pan-tilt.c`) is included to demonstrate how to program the pan-tilt camera. You are invited to study it and use elements of it in your own applications.

For more information on available C functions for the Kameleon or the REB, please see their respective documentation available from the K-Team website (http://www.k-team.com). For general information on how to compile your C program and download to the Kameleon, please see the GNU cross-compiler documentation also available from the K-Team website.

3. Using a Koala

If you are using the camera with the Koala robot, you must:

- Securely mount the Kameleon to the robot by fixing it under the removable cover to the four attachment points labeled number 2 in figure 3a. If you are also using a PC-104 onboard the Koala, you need to mount the Kameleon on top of the PC-104 by using spacers (you will no longer be able to use the Koala cover).
- Mount the camera to the Koala accessory deck using its M3 mounting hole. Note that if you mount two cameras you must be careful to mount them such that the brass gears of each camera cannot touch each other! Recommended mounting holes are (2,3) and (12,3) when viewing the Koala from the front, where the front leftmost hole is (1,1).
- Connect the Kameleon to the Koala's MMA port, marked as number 7 in figure 3a, using the MMA ribbon cable. If you are using a PC-104 as well, you will need to connect it to the Koala and Kameleon using the provided 3-way MMA ribbon cable.
- Connect the 2-pin REB power connector to the power supply connector number 5 shown in figure 3b.
- You do not need to connect the 8-pin AMP connector to the Kameleon—it is powered via the MMA cable.



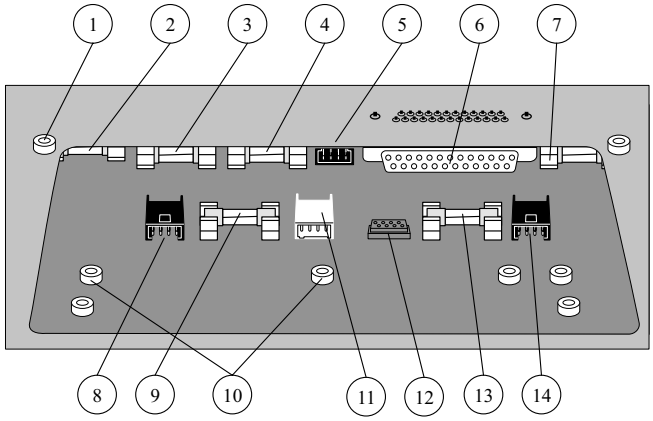
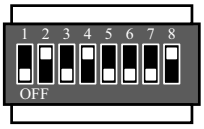


Figure 3a (left) – MMA port on Koala (number 7)
Figure 3b (right) – Power connectors inside the Koala (numbers 5, 8 and 14)

You may either command the camera from a simple ASCII terminal via the serial port, or write your own custom C program. It is recommended you first test the camera from a terminal before writing your custom C program.

3.1. Using a Terminal

First you must set the Kameleon into MMA mode by adjusting the DIP switches located near the power connector as shown in figure 4.



SerCom communication on MMA, MMA slave 0

Figure 4 – DIP switch setting for MMA mode

Connect to the Koala with a terminal program as you normally would to command the Koala. Now open a dedicated MMA channel to the Kameleon by typing: **`REMOTE MMA0`**

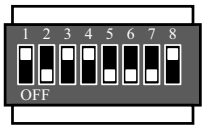
All subsequent commands will be passed directly to the Kameleon SerCom module. You may now initialize and control the camera exactly as described above in section 2.1.

To leave dedicated mode and return to the Koala SerCom type: **`exit`**

3.2. Writing a C program

You may write a custom C program onboard the Koala to control the camera by using the MMA extensions. Simply include the call `mma_reset()` at the beginning of your program, and you will then be able to use the command `mma_send_buffer_0()` to send the standard terminal commands over the MMA channel, and `mma_receive_byte_0()` to read the responses. Please see the included `test_MMA.c` example file for more details.

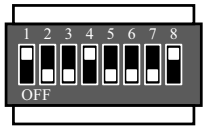
For complete flexibility you may wish to write a custom C program on *both* the Koala and Kameleon. In this case you need to use `mma_reset()` on both sides, and then simply pass text strings back and forth between the two, giving you the flexibility to develop your own communications protocol. In this case you must not deviate SerCom to the MMA channel, but instead leave SerCom to communicate over the serial port using the DIP switch configuration shown in Figure 5.



SerCom communication on RS232 at 38400, MMA slave 0

Figure 5 – DIP Switch setting for bi-directional MMA programs

If you wish your program on the Kameleon to start automatically at boot-up, you must first put your program in flash memory by following the instructions in the Kameleon documentation. You can then tell the Kameleon to automatically load your program in the user flash by choosing the following DIP switch settings shown in Figure 6.



User FLASH application, MMA slave 0

Figure 6 – DIP Switch setting to start the User Flash with MMA

4. Pan-Tilt Command Protocol

On startup, the program in the user-flash of the Kameleon performs a calibration of the cameras and then presents one of two possible menus, depending on whether you have two cameras or not.

One camera

If you have one pan-tilt camera installed, the following self-explanatory menu will appear after calibration. Each command is confirmed by a lowercase character ‘p’ or ‘t’. You may either use the commands interactively, or write a program to use this protocol over the serial port or the MMA port.

```
-------------------------
     PAN-TILT module
-------------------------
P,pos     - move PAN motor (center pos = 0)
P         - print current PAN position
T,pos     - move TILT motor (center pos = 0)
T         - print current TILT position
x         - exit to SerCom (38400 baud)
```

Two cameras

If you have two pan-tilt cameras installed, the following self-explanatory menu will appear after calibration. Each command is confirmed by a lowercase character ‘p’ or ‘t’. You may either use the commands interactively, or write a program to use this protocol over the serial port or the MMA port.

```
-------------------------
     PAN-TILT module
-------------------------
P,cam,pos - move PAN motor (cam = 0 or 1, center pos = 0)
P,cam     - print current PAN position (cam = 0 or 1)
T,cam,pos - move TILT motor (cam = 0 or 1, center pos = 0)
T,cam     - print current TILT position (cam = 0 or 1)
x         - exit to SerCom (38400 baud)
```

5. Software Examples

pan_tilt.c

```
/*
   ; Initialize and control one or two pan-tilt cameras
   ; ================================================

   ;-------------------------------------------------------------------------
   ; Author:    Skye Legon  6/2/01   (slegon@k-team.com)
   ; Modifs:
   ;
   ; This program will initialize one or two pan-tilt cameras connected
   ; to a Kameleon controller board.  It auto-detects whether there are
   ; one or two cameras, and then performs a calibration routine to
   ; find the center positions and limits.
   ;
   ; It then provides a simple serial protocol to command the pan and
   ; tilt axes of the camera(s), suitable for manual testing or for use
   ; with a Koala robot.
   ;
   ; NOTE: the current compiler package does not redirect the output of
   ;       "printf" to the COM port, but instead to the SER port.  However
   ;       in mode MMA only messages written to the COM port are passed
   ;       via MMA to the Koala.  To temporarily get around this, this program
   ;       replaces all "printf" commands with a "Send" function that uses
   ;       the COM port.
   ;
   ;-------------------------------------------------------------------------
 */

#define TEST_SPEED     20
#define TORQUE_OFFSET 11
#define TORQUE_LIMIT   25

#include   <sys/kos.h>
#include   <stdlib.h>
#include   <stdio.h>
#include   <string.h>

int32 _limits[4] = {0,0,0,0};
int32 _pan[2] = {-1,-1};
int32 _tilt[2] = {-1,-1};

void Send(char *string)
{
  // note that com_reserve_channel() does NOT seem to work.  Instead we
  // use com_send_buffer itself to test whether the port is free or not
  while (com_send_buffer(string, strlen(string)) < 0) tim_switch_fast();
}

int32 sgn(int32 num)
{
  return (num < 0 ? -1 : 1);
}

int32 InitializeMotor(uint8 numMotor)
{
  int32 range, speed;
  uint32 torque;

  mot_config_speed_1m(numMotor,150,4,0);
  mot_put_sensors_1m(numMotor,0);

  // turn slowly
  mot_new_speed_1m(numMotor,-TEST_SPEED);
  tim_suspend_task(500);
  // check to see if the motor exists
  // (if it doesn't move AND has low torque, it isn't there!)
  torque = sens_get_ana_value(TORQUE_OFFSET+numMotor);
  if ((mot_get_position(numMotor) == 0) && (torque < TORQUE_LIMIT)) return -1;

  // loop until we hit the stop pin
  do
  {
    tim_suspend_task(100);
```

```
    speed = mot_get_speed(numMotor);
  }
  while (abs(speed) > 0);

  mot_stop_1m(numMotor);
  mot_put_sensors_1m(numMotor,0);
  // move in other direction now
  mot_new_speed_1m(numMotor,TEST_SPEED);
  do
  {
    tim_suspend_task(100);
    speed = mot_get_speed(numMotor);
    fflush(stdout);
  }
  while (abs(speed) > 0);

  range = mot_get_position(numMotor);
  mot_stop_1m(numMotor);
  mot_config_speed_1m(numMotor,200,0,0);
  // go to centre
  mot_new_position_1m(numMotor,(range/2));
  // wait until we've stopped moving (check bit 18 "on-position")
  while ((mot_get_status(numMotor) & (0x01 << 18))==0) tim_switch_fast();
  // set center position to be zero
  mot_put_sensors_1m(numMotor,0);
  mot_new_position_1m(numMotor,0);

  // return limit just a bit inside the stop pins
  return (range/2 - 200);
}

int32 NewMotorPosition(uint8 numMotor, int32 newPosition)
{
  char buffer[80];

  if (numMotor>3) return -1; // invalid motor
  if (_limits[numMotor]==0) return -2; // no motor, or motor not configured
  if (abs(newPosition) > _limits[numMotor]) // out of bounds
  {
    sprintf(buffer, "WARNING: motor position %ld out of bounds...",
      newPosition);
    Send(buffer);
    newPosition = _limits[numMotor] * sgn(newPosition);
    sprintf(buffer, "set to limit %ld.\r\n", newPosition);
    Send(buffer);
  }

  mot_new_position_1m(numMotor, newPosition);

  return 0;
}

void procInit(void *numMotor)
{
  _limits[(uint8)numMotor] = InitializeMotor((uint8)numMotor);

  exit(0);
}

void PrintHelp(uint8 numMotors)
{
  Send("--------------------------                                  \r\n");
  Send("     PAN-TILT module                                       \r\n");
  Send("--------------------------                                  \r\n");
  if (numMotors==2)
  {
    Send("P,pos     - move PAN motor (center pos = 0)               \r\n");
    Send("P         - print current PAN position                    \r\n");
    Send("T,pos     - move TILT motor (center pos = 0)              \r\n");
    Send("T         - print current TILT position                   \r\n");
  }
  else
  {
    Send("P,cam,pos - move PAN motor (cam = 0 or 1, center pos = 0) \r\n");
    Send("P,cam     - print current PAN position (cam = 0 or 1)     \r\n");
    Send("T,cam,pos - move TILT motor (cam = 0 or 1, center pos = 0)\r\n");
    Send("T,cam     - print current TILT position (cam = 0 or 1)    \r\n");
  }
  Send("x         - exit to SerCom (38400 baud)                   \r\n");
```

```
}

int main()
{
  uint8 i=0, numMotor, numMotors=0;
  uint32 taskID[8];
  int32 parms[20];
  int32 receivedByte;
  char userCommand[80];
  char buffer[80];

  ext_reset();
  mot_reset();
  var_reset();
  sens_reset();
  str_reset();

  // configure all 2 or 4 motors simultaneously by launching parallel tasks
  Send("Configuring motors ...\r\n");
  for (numMotor=0; numMotor<4; numMotor++)
  {
    taskID[numMotor] =
      install_task_parms("Init\r\n", 800, procInit, (void *)numMotor);
  }

  // now wait until all four tasks have finished
  for (numMotor=0; numMotor<4; numMotor++)
  {
    Send("-----------------------\r\n");
    sprintf(buffer, "Motor %d...\r\n", numMotor);
    Send(buffer);
    while (_limits[numMotor] == 0) tim_switch_fast();

    if (_limits[numMotor] < 0)
    {
      sprintf(buffer, "No motor found on port %d.  Skipping...\r\n", numMotor);
      Send(buffer);
    }
    else
    {
      numMotors++;
      sprintf(buffer,
        "Motor successfully configured.  Range limit: (+/- %ld) \r\n",
        _limits[numMotor]);
      Send(buffer);
    }
  }

  Send("Finished initialization.\r\n");

  if (numMotors==2) // only one camera
  {
    Send("Found one pan-tilt camera.  Auto-detecting PAN and TILT axes...\r\n");
    for (numMotor=0; numMotor<4; numMotor++)
    {
      if (_limits[numMotor] > 2500)
      {
        sprintf(buffer, "TILT axis is motor %d.\r\n", numMotor);
        Send(buffer);
        _tilt[0] = numMotor;
      }
      else if (_limits[numMotor] > 1500)
      {
        sprintf(buffer, "PAN axis is motor %d.\r\n", numMotor);
        Send(buffer);
        _pan[0] = numMotor;
      }
    }
  }
  else if (numMotors==4) // two cameras
  {
    Send("Found two pan-tilt cameras.  Auto-detecting PAN and TILT axes...\r\n");
    for (numMotor=0; numMotor<4; numMotor++)
    {
      if (_limits[numMotor] > 2500)
      {
        sprintf(buffer, "TILT axis is motor %d.\r\n", numMotor);
        Send(buffer);
        _tilt[numMotor/2] = numMotor;
```

```
    }
    else if (_limits[numMotor] > 1500)
    {
      sprintf(buffer, "PAN axis is motor %d.\r\n", numMotor);
      Send(buffer);
      _pan[numMotor/2] = numMotor;
    }
  }
}
else
{
  // if we find 0, 1, or 3 motors signal an error
  sprintf(buffer,
    "Unrecognized configuration (%d motors).  Exiting.\r\n", numMotors);
  Send(buffer);
  tim_suspend_task(500);
  bios_restart_launch('FUSC', 6);
}

// now start serial command protocol

PrintHelp(numMotors);

// clean buffer
while (com_receive_byte() >= 0) tim_switch_fast();

for(;;)
{
  i = 0;
  do
  {
    if ((receivedByte = com_receive_byte()) > 0)
    {
      // restart SerCom 38400 on 'x'
      if ((char)receivedByte == 'x') bios_restart_launch('FUSC', 6);
      // check for backspace
      else if (((char)receivedByte == '\b')&&(i>0)) i--;
      // accept only "normal" characters
      else if (((char)receivedByte >= ' ')&&((char)receivedByte <= 'z'))
        userCommand[i++] = (char)receivedByte;
    }
    else
      tim_switch_fast();
  }
  while ((char)receivedByte != '\r');

  userCommand[i++] = '\0';

  switch (userCommand[0])
  {
    case 'P':
    {
      if (numMotors==4)
      {
        if(sscanf(userCommand, "P,%ld,%ld", &parms[0], &parms[1]) == 2)
        {
          if ((parms[0]==0)||(parms[0]==1))
          {
            NewMotorPosition(_pan[parms[0]], parms[1]);
            Send("p\r\n");
          }
          else
            Send("Invalid camera (0 or 1).\r\n");
        }
        else if(sscanf(userCommand, "P,%ld", &parms[0]) == 1)
        {
          sprintf(buffer, "p,%ld\r\n", mot_get_position(_pan[parms[0]]));
          Send(buffer);
        }
        else
          Send("Did not understand input.\r\n");
      }
      else
      {
        if(sscanf(userCommand, "P,%ld", &parms[0]) == 1)
        {
          NewMotorPosition(_pan[0], parms[0]);
          Send("p\r\n");
        }
```

```
          else
          {
            sprintf(buffer, "p,%ld\r\n", mot_get_position(_pan[0]));
            Send(buffer);
          }
        }

        break;
      }
      case 'T':
      {
        if (numMotors==4)
        {
          if(sscanf(userCommand, "T,%ld,%ld", &parms[0], &parms[1]) == 2)
          {
            if ((parms[0]==0)||(parms[0]==1))
            {
              NewMotorPosition(_tilt[parms[0]], parms[1]);
              Send("t\r\n");
            }
            else
              Send("Invalid camera (0 or 1).\r\n");
          }
          else if(sscanf(userCommand, "T,%ld", &parms[0]) == 1)
          {
            sprintf(buffer, "t,%ld\r\n", mot_get_position(_tilt[parms[0]]));
            Send(buffer);
          }
          else
            Send("Did not understand input.\r\n");
        }
        else
        {
          if(sscanf(userCommand, "T,%ld", &parms[0]) == 1)
          {
            NewMotorPosition(_tilt[0], parms[0]);
            Send("t\r\n");
          }
          else
          {
            sprintf(buffer, "t,%ld\r\n", mot_get_position(_tilt[0]));
            Send(buffer);
          }
        }

        break;
      }
      default:
      {
        PrintHelp(numMotors);
      }
    }
  }

  return 0;
}
```

test_MMA.c

```
/*
;--------------------------------------------------------------------------
; Author:      Jaquet C.             15/05/2000
;
; Project:      Kameleon376SBC
; Goal:        Test MMA communication between two systems.
;--------------------------------------------------------------------------
*/

#include       <sys/kos.h>
#include       <stdlib.h>
#include       <stdio.h>
#include       <string.h>

/*
 * Process 0 - This process changes the state of the LED 0 every 500ms
 */

static void process_0 ()
{
  for (;;)
    {
      tim_suspend_task (5000);
      var_change_led (0);
    }
}

/*
 * Process 1
 */

#define        kMaxRecBuf     50

void process_1()
{
       uint8  RecBuf[kMaxRecBuf];
       uint8  *SndBuf;
       uint8  i = 0;

       uint8  *Cmd;

       uint32 status = 0;

       printf("Test communication between two system via MMA channel \r\n");
       printf("---------------------------------------------------- \r\n\n");
       printf("The Led number 2 on the slave card must blink  every  \r\n");
       printf("two seconds.                                          \r\n");

       RESERVE_MMA0;


       for (;;)
       {
       tim_suspend_task(2000);
       SndBuf = "L,1,2\n\r";

       status = mma_send_buffer_0(SndBuf, strlen(SndBuf));

       printf("Message sent : %s\n\r", SndBuf);
              do
              {
                     status = mma_receive_byte_0();
                     if (status != -1)
                     {
                            RecBuf[i++] = (char)status;

                            if (((char)status != '\n') & ((char)status != '\r'))
                              printf("Received char(s) %2li . . . %c \r\n", i, (char)status);
                            else
                              printf("Received char(s) %2li . . . CR or LF\r\n");

                     }
              } while (status != '\n');
              RecBuf[i] = '\0';
              printf("\nComplete buffer is : %s \r\n", RecBuf);

       i = 0;
```

```
        }
        RELEASE_MMA0;
        kill_task(tim_get_id());
}

/*
 * MAIN
 *
 * - Initialise the used managers: tim & bios are initialised at the start-up.
 * - Launch all the processes.
 * - Kill the "main". At this moment only the launched processes are executed.
 */

void main(void)
{
        uint32 vIDProcess[3];
        int32  status;

        static char    prName_0[] = "User 0 process, EF-98     Rev. 1.00\r\n";
        static char    prName_1[] = "User 1 process, EF-98     Rev. 1.00\r\n";

        com_reset();
        var_reset();
        mot_reset();
        tim_reset();
        mma_reset();

        status = install_task(prName_0, 800, process_0);
        if (status == -1) exit(0);
        vIDProcess[0] = (uint32)status;

        status = install_task(prName_1, 800, process_1);
        if (status == -1) exit(0);
        vIDProcess[1] = (uint32)status;
}
```