

## Appendix 1 - Arduino Programming of Automatic Solar Panel Tilt

Six functions via three buttons:

1. Activate automatic tracking: short press the center button.
2. Place the solar panel horizontally: press the central button and hold it down for more than 2 seconds.
3. Move the solar panel to the left and deactivate automatic tracking: press and hold the left button. The solar panel will automatically stop at maximum. left corner.
4. Move the solar panel to the right and turn off automatic tracking: long press the right button. will automatically stop at max. right angle.
5. Calibrate (balance) the photoresistors: press and hold the left and right buttons simultaneously. Calibration is stored in EEPROM (Electrically Erasable Programmable Read-Only Memory), so it is not lost on power down. To calibrate, center the solar panel, and place the tricycle in homogeneous light, i.e. under a cloudless gray sky with no sun, or just under the sun (the tricycle should point towards the sun) . Sensor calibration is done by measuring the difference in output of the two sensors, and scaling the output on one of the sensors so that both sensors generate the same output when calibrated.
6. Calibrate Auto-Leveling: Simultaneously press and hold the left and center buttons to save the current light level under which auto-leveling should occur. Calibration is stored in EEPROM, so it is not lost on power down. The current light level is measured by adding the output of the two photoresistors together dividing by two. To calibrate auto-leveling, center the solar panel and place the trike in even light that is too dark for meaningful auto-tracking, i.e. under gray skies with no clouds or sun.

All thanks to this code:

```
// Solar panel control and tracking feature on one axis.
// - Linear Actuator is used to move the the solar panel
// - Motor controller used to control Linear Actuator
// - Accerometer MMA8451 used to determine panel angle
// - 2 x photo resistors at the front used to determine optimal position
// - Center button used to switch on auto tracking and centering (Default press for longer that 1 Second)
// - Left/right button used to move solar panel while held down - switches off auto tracking
// - Left/right buttons also used when hold-pressed simultaneously to calibrate LDR's - switches off auto tracking -
//   set panel to center position, ensure same light intensity to both LDR's, press left and right buttons at the samtime for a couple of
seconds
// - Left/center buttons also used when hold-pressed simultaneously to calibrate min light - switches off auto tracking -
//   set panel to center position, ensure same light intensity to both LDR's, press left and center buttons at the samtime for a couple of
seconds
// - When powering up, solar tracker is in manual mode
// - When too dark and auto tracking enabled, solar panel is centered
// - Panel/cktuator stops in the extreme angles +/- 45 degrees
// - Response time for autotracking moves = TIME_LDR_READ * LDR_COUNT (Default 1 Second)
// - Response time for panel angle measurement = TIME_ANGLE_READ * ANGLE_COUNT (Default 0.5 Second)
// ----- SCENARIOS -----
#define DEBUG_CENTER // define this to debug centering
#define DEBUG // define this to debug
#define GILLES // Define this if compiled for Gilles trike
// ----- CONFIGURATION -----
#define ANGLE_HYST 0 // When centering, +/- ANGLE_HYST degrees stop centering
#define CENTER_HOLD 1000 // Number of milliseconds to hold center button to activate centering.
#define LIGHT_HYST 5 // In percent: When tracking, right/left diff less than LIGHT_HYST% of right/left value of stop tracker.
#define TIME_ANGLE_READ 10 // Time between angle measurements in milliseconds
#define TIME_LDR_READ 100 // Time between light measurements in milliseconds
#define LDR_COUNT 10 // Number of light measurements for moving averages
#define LIGHT_MIN 200 // If light more than LIGHT_MIN, panel will auto center - this is initial value
```

```

#define ANGLE_COUNT 50 // Number of angle measurements for moving averages
// -----
#include <EEPROM.h>
#include <Wire.h>
#include <Adafruit_MMA8451.h>
#include <Adafruit_Sensor.h>
#include <movingAvg.h>
#define MOVE_CENTER 1
#define MOVE_RIGHT 2
#define MOVE_LEFT 3
#define MOVE_STOP 0
#define MOVE_INIT 4
#define BUT_RELEASED 3
#define BUT_HOLD 2
#define BUT_PRESSED 1
#define BUT_IDLE 0
#define LEFT 1
#define RIGHT 2
#define STOP 0
#define ERR - 1
#ifdef GILLES
#define Center_Button_Pin 2
#define Left_Button_Pin 7
#define ANGLE_CALIBRATION -1 // Offset added to angle measurement
#endif
#define Right_Button_Pin 8
#define Ldr_Left_Pin A0
#define Ldr_Right_Pin A1
#define I2C_Data_Pin 4
#define I2C_Clock_Pin 5
#define MMA8451_Addr 0X1D
#define IN1_Pin 9
#define IN2_Pin 10
Adafruit_MMA8451 mma = Adafruit_MMA8451();
bool flag_center = true; // true if accelerometer is running
bool flag_auto = false; // true if auto tracking is enabled
bool flag_auto_last = false;
bool panel_centered = false; // true if panel has been centered
bool too_dark = false;
int panel_move = MOVE_INIT;
int panel_move_last = MOVE_INIT;
int motor_stat = STOP;
int min_light = LIGHT_MIN; // min light
movingAvg ldr_right(LDR_COUNT);
movingAvg ldr_left(LDR_COUNT);
movingAvg panel_angle(ANGLE_COUNT);
float calibration;
int threshold;
int tracker_angle_value = 0;
int tracker_angle_last = 0;
volatile int left_but = HIGH;
volatile int right_but = HIGH;
volatile int center_but = HIGH;
volatile int left_but_last = HIGH;
volatile int right_but_last = HIGH;
volatile int center_but_last = HIGH;
volatile int left_change = false;
volatile int right_change = false;
volatile int center_change = false;
volatile int left_hold = false;
volatile int right_hold = false;
volatile int center_hold = false;
volatile int left_wait = false;
volatile int right_wait = false;
volatile int center_wait = false;
unsigned long ldr_start;
unsigned long ldr_now;
unsigned long angle_start;
unsigned long angle_now;
unsigned long time_center = 0;
// Function declarations
int solar_panel_angle(int);
int left_button();
int right_button();
int center_button();
int sun_tracker(int);

```

```

int linear_aktuator(int);
int read_accerometer();
int solar_panel_move(int);
void ldr_calibrate ( void );
void min_light_calibrate (void);
// the setup function runs once when you press reset or power the board
void setup() {
  ldr_right.begin();
  ldr_left.begin();
  panel_angle.begin();
  Serial.begin(115200);
  Serial.println("");
  Serial.println("VELOKS Tracker Initializing");
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR1B |= (1 << CS10); // prescaler 1
  TIMSK1 |= (1 >> TOIE1); // enable timer overflow
  Serial.println("MMA8451 Start");
  if (!mma.begin(MMA8451_Addr)) {
    Serial.println("MMA8451 Couldnt start");
    flag_center = false;
  } else {
    Serial.print("MMA8451 Started! - ");
    mma.setRange(MMA8451_RANGE_2_G);
    Serial.print("Range = ");
    Serial.print(2 << mma.getRange());
    Serial.println("G");
    flag_center = true;
  }
  pinMode(Center_Button_Pin, INPUT_PULLUP);
  pinMode(Left_Button_Pin, INPUT_PULLUP);
  pinMode(Right_Button_Pin, INPUT_PULLUP);
  pinMode(IN1_Pin, OUTPUT); // Aktuator Control
  pinMode(IN2_Pin, OUTPUT); // Aktuator Control
  ldr_start = millis();
  ldr_now = ldr_start;
  angle_start = millis();
  angle_now = angle_start;

  EEPROM.get (10, min_light);
  if (min_light == 0 ) { // Min_light when started first time
    min_light = LIGHT_MIN;
    EEPROM.put ( 10, min_light);
  }
  EEPROM.get (0, calibration);
  if (calibration == 0 ) { // Calibration factor must be 1 when started first time
    calibration = 1.0;
    EEPROM.put ( 0, calibration);
  }
  Serial.print("LDR Calibration = ");
  Serial.println(calibration);
  Serial.print("Min Light Calibration = ");
  Serial.println(min_light);

  Serial.println("VELOKS Tracker Started");
  Serial.println("");
}
// Timer interrupt function that reads the 3 button values
ISR(TIMER1_OVF_vect) {
  left_but = digitalRead(Left_Button_Pin);
  right_but = digitalRead(Right_Button_Pin);
  center_but = digitalRead(Center_Button_Pin);
  if (left_but != left_but_last) {
    left_but_last = left_but;
    left_change = true;
  }
  if (right_but != right_but_last) {
    right_but_last = right_but;
    right_change = true;
  }
  if (center_but != center_but_last) {
    center_but_last = center_but;
    center_change = true;
  }
}
}

```

```

void loop() { // Main loop
  int lb;
  int rb;
  int cb;
  panel_move = solar_panel_move(panel_move); // Move panel if needed
  if (flag_auto) panel_move = sun_tracker(panel_move); // If autotracking enabled, determine if panel must move
  // Read the status of the 3 buttons
  lb = left_button();
  rb = right_button();
  cb = center_button();
  if ( lb == BUT_HOLD && rb == BUT_HOLD && cb != BUT_HOLD ) { // Calibrate LDR
    panel_move=solar_panel_move(MOVE_STOP);
    flag_auto = false;
    ldr_calibrate ();
  }
  else if ( lb == BUT_HOLD && cb == BUT_HOLD ) { // Calibrate Min Light
    panel_move=solar_panel_move(MOVE_STOP);
    flag_auto = false;
    min_light_calibrate ();
  }
  else {
    if (cb == BUT_PRESSED && panel_move != MOVE_CENTER) {
      flag_auto = true;
    }
    if (lb == BUT_HOLD && rb != BUT_HOLD) {
      panel_move = MOVE_LEFT;
      too_dark = false;
      flag_auto = false;
    } // Move left and enter manual mode
    if (rb == BUT_HOLD && lb != BUT_HOLD) {
      panel_move = MOVE_RIGHT;
      too_dark = false;
      flag_auto = false;
    } // Move right and enter manual mode
    if (cb == BUT_HOLD) {
      panel_move = MOVE_CENTER;
      panel_centered = false;
      flag_auto = false;;
    } // Center panel Move left and enter manual mode
    if (rb == BUT_IDLE &&
        lb == BUT_IDLE &&
        panel_move != MOVE_CENTER &&
        flag_auto == false) panel_move = MOVE_STOP; // Stop moving when left/right buttons are not pressed
  }
}

// Function calibrates LDR's so that they read equal - must be equal light to both LDR's when executed - e.g panel level gray sky
void ldr_calibrate ( void ) {
  float ldr_right_value;
  float ldr_left_value;
  ldr_right_value = analogRead(Ldr_Right_Pin);
  ldr_left_value = analogRead(Ldr_Left_Pin);
  calibration = ldr_right_value / ldr_left_value;
  EEPROM.put ( 0, calibration);
  Serial.print("LDR Calibration = ");
  Serial.println(calibration);
  delay ( 5000 );
}

// Function reads LDR values and determines min light for sun tracking
void min_light_calibrate (void) {
  int ldr_right_value;
  int ldr_left_value;
  ldr_right_value = analogRead(Ldr_Right_Pin);
  ldr_left_value = analogRead(Ldr_Left_Pin);
  min_light = (ldr_right_value + ldr_left_value * calibration)/2;
  EEPROM.put ( 10, min_light);
  Serial.print("Min Light Calibration = ");
  Serial.println(min_light);
  delay ( 5000 );
}

// Function reads LDR values and determines if solar panel must move and in which direction
int sun_tracker(int command) {
  int ldr_left_value;
  int ldr_right_value;
  int ret = command;
  int diff;
  // Exit id less than TIME_LDR_READ milliseconds threshold past

```

```

ldr_now = millis();
if ((ldr_now - ldr_start) < TIME_LDR_READ) return (ret);
ldr_start = millis();
ldr_left_value = ldr_left.reading( (int) analogRead( Ldr_Left_Pin)* calibration );
ldr_right_value = ldr_right.reading(analogRead(Ldr_Right_Pin));
diff = ldr_left_value - ldr_right_value;
threshold = (ldr_left_value + ldr_right_value) / 100 * LIGHT_HYST;
#ifdef DEBUG
Serial.print("Calibration=");
Serial.print(calibration);
Serial.print(" Threshold=");
Serial.print(threshold);
Serial.print(" Diff=");
Serial.print(diff);
Serial.print(" LDR-L=");
Serial.print(ldr_left_value);
Serial.print(" LDR-R=");
Serial.println(ldr_right_value);
#endif
if (ldr_left_value > min_light && ldr_right_value > min_light) { // If too little light
if (too_dark == false) {
#ifdef DEBUG
Serial.println("TRACKING - CENTERING");
#endif
ret = MOVE_CENTER; // Center the panel
panel_centered = false;
too_dark = true;
return (ret);
}
} else if ((diff + threshold) < 0) {
ret = MOVE_RIGHT;
too_dark = false;
#ifdef DEBUG
Serial.println("TRACKING RIGHT");
#endif
} else if ((diff - threshold) > 0) {
too_dark = false;
ret = MOVE_LEFT;
#ifdef DEBUG
Serial.println("TRACKING LEFT");
#endif
} else {
#ifdef DEBUG
Serial.println("TRACKING FOUND");
#endif
ret = MOVE_STOP;
}
return (ret);
}

int linear_aktuator(int direction) { // Function controls Linear Aktuator
int ret = ERR;
if (direction == STOP && motor_stat != STOP) {
digitalWrite(IN1_Pin, LOW);
analogWrite(IN2_Pin, 0);
ret = STOP;
} else if (direction == LEFT) {
digitalWrite(IN1_Pin, LOW);
analogWrite(IN2_Pin, 255);
ret = LEFT;
} else if (direction == RIGHT) {
digitalWrite(IN2_Pin, LOW);
analogWrite(IN1_Pin, 255);
ret = RIGHT;
}
return (ret);
}

int read_accrometer() { // Function calculates solarpanel incline angle
int x, y, z; //three axis acceleration data
double roll = 0.00, pitch = 0.00; //Roll & Pitch are the angles which rotate by the axis X and y
double x_Buff;
double y_Buff;
double z_Buff;
if (flag_center == true) {
mma.read();
x = mma.x;
y = mma.y;
}

```

```

z = mma.z;
x_Buff = float(x);
y_Buff = float(y);
z_Buff = float(z);
roll = atan2(y_Buff, z_Buff) * 57.3;
pitch = atan2((-x_Buff), sqrt(y_Buff * y_Buff + z_Buff * z_Buff)) * 57.3;
if (roll > 0) roll = roll - 180;
else if (roll < 0) roll = roll + 180;
roll = panel_angle.reading(roll) + ANGLE_CALIBRATION;
} else {
roll = 0; // Assume centered if accelerometer is not started.
}
return (roll);
}
int solar_panel_angle(int command) { // Function returns solar panel incline angle with hysteresis.
angle_now = millis();
if ((angle_now - angle_start) < TIME_ANGLE_READ)
return (tracker_angle_value);
angle_start = millis();
if (command != panel_move_last) { // Update last command and print new state
Serial.print("COMMAND = ");
switch (command) {
case MOVE_LEFT:
Serial.println("MOVE_LEFT");
break;
case MOVE_RIGHT:
Serial.println("MOVE_RIGHT");
break;
case MOVE_STOP:
Serial.println("MOVE_STOP");
break;
case MOVE_CENTER:
Serial.println("MOVE_CENTER");
break;
case MOVE_INIT:
Serial.println("MOVE_INIT");
break;
defaults: Serial.println("Illegal Command");
break;
}
panel_move_last = command;
}
tracker_angle_value = read_accrometer(); // get roll angle in degrees
return (tracker_angle_value);
}
int solar_panel_move(int move) { // Function reads solar panel angle and moves panel depending upon current state.
int result = move;
int panel_angle = solar_panel_angle(move);
if (move == MOVE_CENTER && panel_centered == false) {
if (panel_angle > 0 + ANGLE_HYST) {
motor_stat = linear_aktuator(RIGHT);
#ifdef DEBUG_CENTER
Serial.print("CENTER RIGHT, ANGLE = ");
Serial.println(panel_angle);
#endif
} else if (panel_angle < 0 - ANGLE_HYST) {
motor_stat = linear_aktuator(LEFT);
#ifdef DEBUG_CENTER
Serial.print("CENTER LEFT, ANGLE = ");
Serial.println(panel_angle);
#endif
} else {
#ifdef DEBUG_CENTER
Serial.print("CENTER FOUND, ANGLE = ");
Serial.println(panel_angle);
#endif
motor_stat = linear_aktuator(STOP);
result = MOVE_STOP;
panel_centered = true;
}
} else if (move == MOVE_RIGHT) {
panel_centered = false;
motor_stat = linear_aktuator(RIGHT);
result = MOVE_RIGHT;
} else if (move == MOVE_LEFT) {
panel_centered = false;
}
}

```

```

    motor_stat = linear_aktuator(LEFT);
    result = MOVE_LEFT;
} else if (move == MOVE_STOP) {
    motor_stat = linear_aktuator(STOP);
    result = MOVE_STOP;
} else if (move == MOVE_INIT) {
    tracker_angle_value = 0; // Set current angle to level
    result = MOVE_STOP;
}
}
return result;
}
int center_button() { // Determine status of center button
int value = BUT_IDLE;
if (center_change == true) {
    if (millis() - time_center < CENTER_HOLD &&
        center_but_last == LOW) {
        value = BUT_PRESSED;
    } else if (center_but_last == LOW &&
        center_hold == false &&
        center_wait == false) {
        value = BUT_PRESSED;
        time_center = millis();
        center_wait = true;
    } else if (center_but_last == LOW && center_wait == true) {
        center_hold = true;
        value = BUT_HOLD;
    } else if (center_but_last == HIGH) {
        value = BUT_RELEASED;
        time_center = 0;
        center_change = false;
        center_hold = false;
        center_wait = false;
    }
}
}
return (value);
}
int right_button() { // Determine status of right button
int value = BUT_IDLE;

if (right_change == true) {
    if (right_but_last == LOW && right_hold == false) {
        value = BUT_PRESSED;
        right_hold = true;
    } else if (right_but_last == LOW && right_hold == true) {
        value = BUT_HOLD;
    } else if (right_but_last == HIGH) {
        value = BUT_RELEASED;
        right_change = false;
        right_hold = false;
    }
}
}
return (value);
}
int left_button() { // Determine status of left button
int value = BUT_IDLE;
if (left_change == true) {
    if (left_but_last == LOW && left_hold == false) {
        value = BUT_PRESSED;
        left_hold = true;
    } else if (left_but_last == LOW && left_hold == true) {
        value = BUT_HOLD;
    } else if (left_but_last == HIGH) {
        value = BUT_RELEASED;
        left_change = false;
        left_hold = false;
    }
}
}
return (value);
}

```