CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Radio Engineering

Systém pro snímání panoramatických fotografií s vysokým rozlišením

System for acquisition of panoramic photos with high resolution

Study programme: Communications, Multimedia and Electronics
Field of study: Multimedia Technology

Supervisor: Ing. Karel Fliegel, Ph.D.

Bc. Jan Štemberg

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra radioelektroniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Jan Štemberg**

Studijní program: Komunikace, multimédia a elektronika
Obor: Multimediální technika

Název tématu: **Systém pro snímání panoramatických fotografií s vysokým rozlišením**

Pokyny pro vypracování:

S využitím motorizované stativové hlavy realizujte automatizovaný systém pro snímání panoramatických fotografií s vysokým rozlišením. Podejte přehled metod pro následné zpracování sejmutého obrazu. Implementujte programové vybavení pro ovládání stativové hlavy a zpracování výsledných fotografií a ověřte účinnost implementovaných algoritmů.

Seznam odborné literatury:

[1] Gonzalez, R. C.: Digital image processing, Pearson, 2002.
[2] Šonka, M., Hlaváč, V., Boyle, R.: Image processing, analysis, and machine vision, Thomson, 2008.

Vedoucí: Ing. Karel Fliegel, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Mgr. Petr Páta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 2. 2015

Abstract

The thesis is focused on design of the panorama and high resolution photography system. First part describes existing solutions, reasons for building own system, requirements for such a system and theory of image stitching methods. Practical part describes mechanical construction and electronic design of the devices. It also deals with implementation of the control software for the microcontrollers in the devices, basic use of stitching algorithm and Android application. In conclusion, panorama system is analyzed, stitching algorithms are realized and compared to open-source and commercial stitching software.

Keywords: hi-resolution photography, panorama photography, motorized tripod head, camera positioning, panorama stitching

Abstrakt

Práce je zaměřená na návrh systému pro tvorbu panoramatických fotografií ve vysokém rozlišení. První část popisuje existující řešení, důvody pro vývoj vlastního systému, požadavky pro takový systém a metody pro skládání obrazu. Praktická část popisuje mechanickou konstrukci a návrh elektroniky pro zařízení. Zabývá se také implementací obslužného softwaru, použití metod pro skládání obrazu a vývojem ovládací aplikace pro Android. V závěru je analyzován systém pro snímání a porovnány nástroje pro skládání panoramat.

Klíčová slova: fotografie s vysokým rozlišením, panoramatická fotografie, motorizovaná stativová hlava, skládání obrazu

Čestné prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Datum: 7.5.2015

……
podpis diplomanta

Acknowledgments

I would like to express the deepest appreciation to the supervisor of my diploma thesis Ing. Karel Fliegel, Ph.D. for his professional approach, valuable advices and motivation during our cooperation. I would also like to thank my parents for their moral and material support during my studies.

In addition, a thank to my best friends who were keeping me alive during the long process of work and special thanks to Tereza Vicková who helped me to get panoramic head in NYC.

Content

Introduction

Photography is still very popular. Although most images are taken by mobile devices with integrated cameras today, there are still a lot of users working with high quality photography equipment like DSLR.

Panoramic photography is an expression of virtual reality. It is evolved by perception of the world and it give us strong experience [1]. Panoramic image is also important research direction. It involves precise capture system, image processing, computer graphics and vision, and many other technologies.

Panorama photography may be gotten by using of special photographic equipment, but these devices are often very expensive and complex. Another way is to use the technique of merging many shifted pictures to one panorama photography.

If we want to take three or five images, it is not a problem. But it is getting worse if a lot of pictures is captured - for example 20 photos horizontally and 4 vertically. It means to take 80 images. And if we want to take these shots in night with long exposure time, manual positioning and capturing of each frame becomes impossible. This is the situation when automation panorama photography system is useful.

As a student of multimedia technology at the Czech Technical University in Prague, I am interested (except sound engineering and other many activities) mainly in photography and image processing. Design of this system is based on long term experience with manual panorama imaging and the need for some automation solution. One of the goal is utilization of the system in practice.

The aim of the project is to design and build automatic panorama photography system that will be precise, remote controlled and simple to use.

The thesis is structured in three chapters. The first gives basic system description and overview about image stitching methods. The second chapter describes mechanical and electronic design of panoramic head and control unit as well as software implementation. It also presents Android application and image stitching software. Last chapter summarizes the results of the work. Each chapter is divided into sections and two subsections. Last level of this structure is numbered just by letters *A, B*…

Chapter I Theory & Task Analysis

This chapter gives an overview about basic description of system, and theory of image stitching. Section 1 describes alternative commercial solution for panorama photography. The second section summarizes requirements of the system and the third explains principles and algorithms for image stitching. It describes process of generation panoramic model, feature detection methods, feature matching and image blending.

1 Existing Solutions

There are some commercial solutions for panorama photography on the market, but most of them are very expensive professional solutions and sometimes does not afford remote connectivity and freedom of control. The possibilities of modification of these commercial systems are quite problematic so it was decided to build own system. The second reason for building own system is much better understanding all features and components of the system. This project should also extend conventional solutions with Bluetooth communication, joystick control, very long operation time, high movement precision and freedom of control by any devices. The best known commercial products are GigaPan High-Resolution system [2] or Syrp Genie Motion Control Time Lapse Device [3].

2 System Requirements

The system should serve as a complex tool for panorama and high resolution photography. The core of the system is motorized tripod head with control mechanism. Although there are some panoramic camera projects, high resolution photography is usually realized with many low resolution images that are stitched and merged together. Images are often captured with lenses with long focal length (300 – 600 mm). Overlaps between each neighboring frames are also required for following stitching process. These facts impose requirements for high accuracy and precision of positioning system. System should also have appropriate range of movement in vertical and horizontal direction. Each set of shots include settings of many parameters such as number of shots in x and y axis, their positions, overlaps etc. Remote control system is required for this purpose and for easy practical use. The process of panorama creating is illustrated on Figure 1.

Figure 1: Process of creating high-resolution panorama photography

2.1 Motion Precision

Precision requirements are based on angle of view of the camera. System is mainly designed for use with DSLR[1] with interchangeable lenses. If we use a lens with focal length 300 mm with APS-C

[1] DSLR - Digital Single-Lens Reflex camera

sensor we can calculate the angle of view. Nikon APS-C[2] sensor (Nikon DX) has dimensions approximately 24 x 16 mm. The configuration is sketched on Figure 2.

The angle of view can be calculated by formula (1)

$$\alpha = 2 \cdot \tan^{-1} \frac{d}{2 \cdot f} \tag{1}$$

Where α is angle of view, d is sensor dimension and f is focal length of used lens

For horizontal direction (2)

$$\alpha_h = 2 \cdot \tan^{-1} \frac{24}{2 \cdot 300} \cong 4.58° \tag{2}$$

For vertical direction (3)

$$\alpha_v = 2 \cdot \tan^{-1} \frac{16}{2 \cdot 300} \cong 3.06° \tag{3}$$

If we want frame overlaps $\frac{1}{5}$ of picture size then we need motion precision $\frac{3.06°}{5} \cong 0.6°$ in vertical direction and $\frac{4.58°}{5} \cong 0.9°$ in horizontal direction [4]. This will affect the choice of used mechanical components and position detection system.

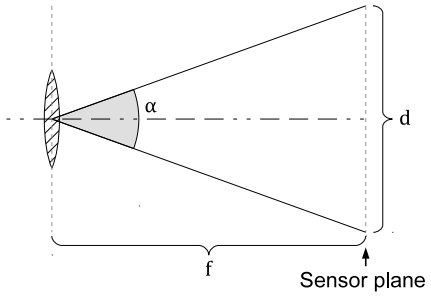


Figure 2: Angle of view of the camera; d is sensor dimension, f is focal length and α is angle of view

2.2 Motion Range

For panorama photography we need relatively wide motion range, especially in horizontal direction. 180° is sufficient range for most panoramas, but system should allow at least 270° horizontal range. Vertical range must does need to be as wide as horizontal and ± 15° from zero position seems to be acceptable for most situations.

2.3 Motion Speed

Speed of the motion system is not so critical for our purposes because it is supposed that mostly static scenes will be captured. More important is spatial reproducibility and stability when head is not moving. System must be robust enough to keep stable position for various types of camera and must minimize vibrations caused by wind or other conditions.

[2] APS-C - Advanced Photo System type-C

2.4 Dimensions, Mobility, Operating Conditions

System must be simply portable and compatible with most of cameras and tripods. That is solved by ¼" fixing screw for both tripod and camera which is universal for most devices. System should also have long operating time with battery power which is needed for long time expositions with a lot of pictures. System must be able to work in wide temperature range. That means range from -20° to +40° to cover cold winter nights and hot summers on direct sunlight.

2.5 Control System

The device requires user friendly system that will ensure control of all head's functions. It will be accessible form PC, smartphone or tablet. For basic use, cable communication is good solution but for outdoor situations remote control by smartphone is better. This feature should provide complete panorama settings as motion range, number of shots in each position with different exposure settings for HDR[3], frame overlap size etc.

3 Panoramic Image Stitching Algorithm

This chapter describes important methods for panorama stitching and generation of panoramic model

3.1 The Generation of Panoramic Model

Process of generation of panorama consists of several steps [1], see Figure 3.

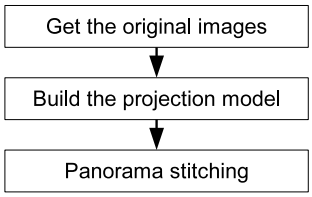


Figure 3: Process of panorama generation

3.1.1 Getting Original Images

An ordinary camera may be used to obtain the pictures, but there are two points that we should keep in mind to make sure that the panorama will be captured correctly. Each neighboring images must have same overlapping regions and images should form the matrix which allows simple composition of the panorama.

3.1.2 Applying of Projection Model

There are three different projection models: cube, sphere and cylinder. Each of them has specific properties and gives different results.

Cylindrical projection model is usually used in most solutions, because it is easy to capture original images and cylindrical surface can be simply expanded to rectangular plane. It allows to map up to 360° and cylinder's core is the viewpoint [1].

[3] HDR - High Dynamic Range photography

A. The Cylindrical Projection

We have to find the corresponding projective point by solving parameter equation of plane image and cylinder:

$$x' = f \cdot arctan\left(\frac{x - \frac{w}{2}}{f}\right) + f \cdot arctan\left(\frac{w}{2f}\right)$$
$$y' = \frac{f \cdot (y - \frac{h}{2})}{\sqrt{\left(x - \frac{w}{2}\right)^2 + f^2}} + \frac{h}{2} \quad (4)$$

where w, h is width and height of plane image, f is the camera's focal length and x' and y' are the corresponding points on cylinder.

3.1.3 Image Stitching

Critical part of image stitching is image registration. The performance of registration algorithm mostly influences photorealistic quality. There are three principles of this process: region-based algorithm, feature-based algorithm and phase correlation algorithm [1].

Region-based algorithm selects a window in one image and compares it with the images waiting for registration. This method has big disadvantage because it is not suitable for images with different perspective or larger rotation. Phase detection can align translational images accurately and the algorithm is independent of scene. Feature-based algorithm selects features in the images and does the feature matching according to the principle of conformity. Advantage of this method is a small computational complexity and good robustness to contrast degree and illumination change. The key problem of this method is to select the features. There are two major algorithms SIFT[4] and SURF[5] for feature detection. There are slightly different ways of detecting features between SIFT and SURF. SIFT builds an image pyramids, filtering each layer with Gaussians while increasing their sigma values and taking the difference. On the other hand, SURF creates "stack" without 2:1 down sampling for higher levels in the pyramid, resulting in the images of the same resolution. Due to the use of integral images, SURF filters the stack using a box filter approximation of second-order Gaussian partial derivatives. [5]

3.2 Image Stitching

Series of images is based on their overlapping regions which are the parts with common information about the scene. Feature-based algorithm extracts some features from the image and then does the point matching. This is followed by removing mismatch and image registration based on feature points. At the end, images are aligned to the panorama and image blending is applied.

3.2.1 Feature Extraction

The way how to select the feature points is the key factor for feature-based image stitching algorithm. The points must characterize the image and they need to satisfy two conditions [1].

a. Feature points of the scene with different illumination conditions, perspective and viewpoint must be just the same

[4] SIFT – Scale Invariant Feature Transform
[5] SURF – Speeded Up Robust Features

b. The points must have sufficient information to match each other

Harris corner algorithm is widely used efficient operator that can quickly and accurately find the points. It has rotation invariance, but it does not have scale invariance. That means that the same object with the different sizes in different image will have different feature information. To solve this, multi-scale Harris corner detection algorithm must be used.

A. Harris Corner Detection Algorithm

The algorithm is based on the change of gray level described by first-order partial derivative. It proposes the matrix M, which is associated with the autocorrelation function of the image. By calculating of the eigenvalue of M, we can determine whether a point is a cornet or not. The sum of gray deviation of regional image can be described with the autocorrelation function as follows [6]:

$$\boldsymbol{E(u,v) = \sum_{x,y} w(x,y)[f(x+u, y+v) - f(x,y)]^2} \tag{5}$$

where $w(x,y)$ is a Gaussian filter. E's Taylor expression is as followed:

$$\boldsymbol{E(u,v) \cong [u\ v] M \begin{bmatrix} u \\ v \end{bmatrix}} \tag{6}$$

In (6), M is 2×2 symmetric matrix:

$$\boldsymbol{M = \begin{pmatrix} \left(\frac{\partial I}{\partial x}\right)^2 & \left(\frac{\partial I}{\partial x}\right)\left(\frac{\partial I}{\partial y}\right) \\ \left(\frac{\partial I}{\partial x}\right)\left(\frac{\partial I}{\partial y}\right) & \left(\frac{\partial I}{\partial y}\right)^2 \end{pmatrix}} \tag{7}$$

where $I(x,y)$ is the gray value, $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ are the image's first order partial derivatives on x-axis and y-axis.

M's eigenvalue can be used to approximate the curvature extreme value of image gray autocorrelation function on one point. If both of the two values are large, that means curvature extreme values on x-axis and y-axis are large. If this occurs, small movement will cause great changes in gray value and that point is marked as a corner.

The determinant of M is in direct relation with product of curvature extreme values. Harris corner detector can be expressed as:

$$\boldsymbol{R = detM - k(traceM)^2} \tag{8}$$

where $detM = \lambda_1 \cdot \lambda_2$, $traceM = \lambda_1 + \lambda_2$, λ_1 and λ_2 are eigenvalues of M, $0.04 \leq k \leq 0.06$.

There are three situations which may occur. When $|R|$ is small, the point is in the smooth region of the image. When $|R|$ is large and $R < 0$, the point is in the edge region. Only when $|R|$ is big and $R > 0$, the point is corner point. Threshold T is usually defined, and if $R > 0$, the point is a corner point.

Harris corner algorithm may be summarized in three steps:

a. Calculation of the first partial derivative on x-axis and y-axis of each point of the image and do Gaussian filter with its square and product.
b. Calculation of R value of each point and if R>T, point is selected as a candidate feature point.
c. Selection of maximum in neighborhood among these candidate points as the final feature point.

B. Multi-scale Harris Corner Detection Algorithm

For practical use, scale invariant is needed. So we adopt multi-scale Harris corner [7] by using a Gaussian image pyramid.

a. We form a Gaussian image pyramid $P_1(x, y)$ using a subsampling rate $s = 2$ and pyramid smoothing width $\sigma_p = 1.0$ for each input image $I(x, y)$.

b. The Harris matrix at level 1 and position (x, y) is the smoothed outer product of the gradients. The output is:

$$\boldsymbol{M_l(x,y) = \nabla_{\sigma_d} P_l(x,y) \nabla_{\sigma_d} P_l(x,y)^T \cdot g_{\sigma_i}(x,y)} \tag{9}$$

The integration scale $\sigma_d = 1.0$ and derivative scale $\sigma_i = 1.5$.

c. Corner strange function may be computed:

$$\boldsymbol{f_{HM}(x,y) = \frac{detM_l(x,y)}{trM_l(x,y)} = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}} \tag{10}$$

λ_1 and λ_2 are eigenvalues of M, $f_{HM}(x, y)$ is also called the harmonic mean of the eigenvalues. We have to find some interesting points. They are located where corner strength function $f_{HM}(x, y)$ is a local maximum in a 3×3 neighborhood, and above the threshold T.

d. We also compute an orientation θ and orientation vector for each point:

$$\boldsymbol{[cos\theta, sin\theta] = u/|u|} \tag{11}$$

u comes from the smoothed local gradient:

$$\boldsymbol{u_l(x,y) = \nabla_{\sigma_o} P_l(x,y)} \tag{12}$$

The integration scale for orientation $\sigma_o = 4.5$. A large derivative scale is desired because the gradient field varies very smoothly across the image.

At the end, we can use adaptive non-maximal suppression (ANMS) strategy [8] to select a fixed number of interest points from each image and make sure, that the points are spatially well distributed over the image.

The result of Harris Corner algorithm is illustrated in Figure 4 [9].

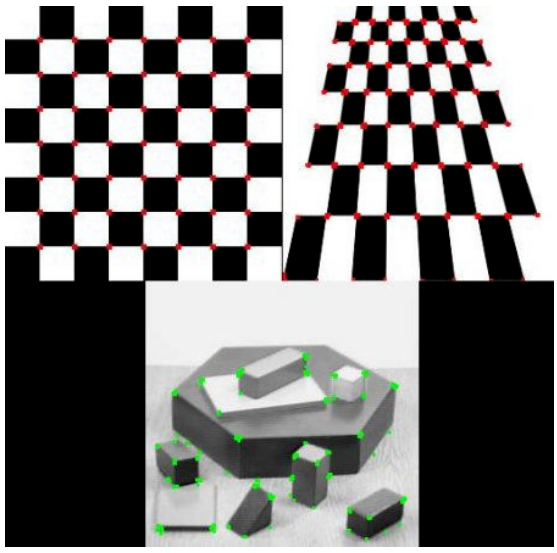


Figure 4: Results of Harris Corner algorithm

C. SURF Feature Detector

The SURF Feature detector is based on applying an approximate Gaussian second derivative mask at many scales. It is more robust to rotation than Harris corner because it applies mask along each axis and at 45° to the axis. The method is very fast because it uses an integral images where the value of a pixel (x,y) is the sum of all values in the rectangle defined by the origin and (x,y). In such an image the sum of the pixels within a rectangle of any size in a source image can be found as the result of 4 operations. This allows a rectangular mask of any size to be applied with very little computing time [10].

We assemble the Hessian matrix:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{L_{xx}} & \boldsymbol{L_{xy}} \\ \boldsymbol{L_{yx}} & \boldsymbol{L_{yy}} \end{bmatrix} \tag{13}$$

where L_{xx} is the convolution of the second derivative of a Gaussian with the image at the point.

The used masks are approximation and they are shown in Figure 6. The crude approximations are valuable because they can be quickly run at any scale with very little computing time. The Hessian determinate values for the same image as Figure 6 are shown in Figure 5 [10].

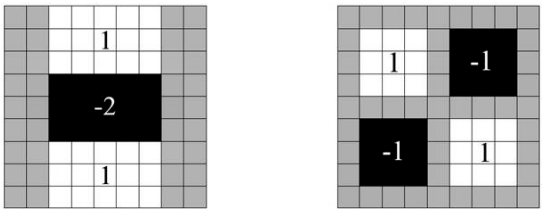


Figure 6: Approximated Gaussian second derivative used for the SURF detector. On the left is the mask used for G_{XX} and G_{YY} and on the right is the mask used for G_{XY}. [10].

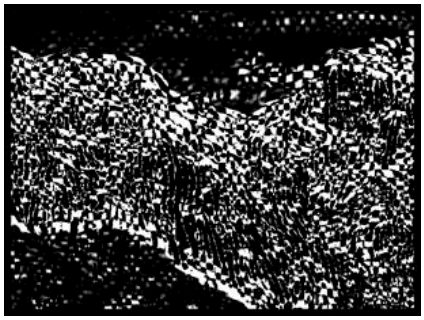


Figure 5: Surf feature values with 15x15 detector size

3.2.2 Feature Matching

We have to find the relation between interest points of two images and this can be done by feature matching. Frequently used method to achieve this is comparing the feature point descriptor's error energy:

$$\boldsymbol{d} = \sqrt{\sum_{\boldsymbol{i=0}}^{\boldsymbol{N}} (\boldsymbol{FV_1(i) - FV_2(i)})^2} \tag{14}$$

Nearest neighbor algorithm [11] is used which is an improvement of k-D tree[6] algorithm [12]. The process begins by creating k-D tree for each image and making sure that the node is the descriptor. Then we will find one image's descriptor in k-D of another image. This repeats until all pairs are matched.

3.2.3 Mismatch Removal

During the feature matching process, some pairs of interest points are not mapped on the same scene (the points have similar descriptor but the scene is different). These points are called exterior points and their pairs would cause problems in parameter calculation process. Therefore, exterior points must be removed. The most widely used algorithm for this reason is RANSAC[7] [13]. The coordinate transformation relation of images can be work out by matching double points. It is defined by the transformation matrix M [1].

$$\begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \\ \boldsymbol{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{m_0} & \boldsymbol{m_1} & \boldsymbol{m_2} \\ \boldsymbol{m_3} & \boldsymbol{m_4} & \boldsymbol{m_5} \\ \boldsymbol{m_6} & \boldsymbol{m_7} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{1} \end{bmatrix} \tag{15}$$

$A' = M \circ A$, A' and A are matching double points.

The RANSAC algorithm can be divided into several steps:

a. Select 4 pairs of matching points from N pairs randomly to establish equations and work out the solution of M's unknown parameters.
b. Calculate the distance between the two points of N-4 pairs. If the distance is smaller than the threshold, the two points are the interior points.
c. Select another 4 pairs and repeat the process.
d. The optimal solution is the parameter with maximal interior points.

3.2.4 Image Registration

In this stage we need to describe the conversion relation between multiple images. For that, finding an appropriate model is the key problem. We cannot use transformation matrix M obtained via RANSAC because projections of the same object in different images will be different since the projection direction is different. The key problem is to eliminate the accumulated error between the images, which complicates the correct image match. This problem can be solved by bundle adjustment [14]. Firstly, we choose one of the images to be a reference surface. Then, each of other images transform to the reference surface, at the end of which all images are on the same surface.

Bundle adjustment process may be described as a reading of each image into the adjustment, and constantly optimizing the parameters of the matrix in the adjustment. Optimizing process includes finding out the best neighbor image for each image and directly calculating the distance between the two neighbor images. Then, matrix between the neighbor images is adjusted by the distance value.

[6] k-D tree – k-Dimensional tree
[7] RANSAC - Random Sample Consensus

In the image I_1, one feature match (u_i, u_j) is given. The value of u_i is also given and is projected to the reference surface and then to its neighbor image. u_i becomes to $\widetilde{u_i}$. The residual between them can be shown as:

$$\boldsymbol{r_{ij} = \left|u_j - \widetilde{u_i}\right| = \left|u_j - H_j^{-1} \cdot H_i \cdot \widetilde{u_i}\right|} \tag{16}$$

where H_i is the matrix transforming image I_i, to the reference surface and H_j^{-1} is the matrix transforming reference surface to the image I_j.

The object function is the sum of all the residual errors shown as equation:

$$\boldsymbol{e = \sum_{i=1}^{n} \sum_{j \in L(i)} \sum_{k \in F(i,j)} \left| f(r_{ij}^{k}) \right|} \tag{17}$$

where $f(i, j)$ is the set of feature matches between image I_i and image I_j. Then, we can update H_i

$$\boldsymbol{H_i = H'_{ij} * H_i} \tag{18}$$

where H'_{ij} is the updating matrix value of image I_1 and it is best neighbor image.

The problem of parameter optimization can be transformed into nonlinear least square problem. This can be implemented as a Levenberg-Marquadt algorithm.

3.2.5 Image Blending

All previous steps realizes geometry stitching which has obvious seam in overlap region. This is caused by different illumination of the same scene on different images.

For image blending, we can use simple and fast average weighted method. The method is illustrated in Figure 7.

In the average weighed blending, the values of features in overlap region are equal to the weighted average values of matching images which can be expressed as [14]:

$$\boldsymbol{p = \frac{d_l}{d_l + d_r} p_l + \frac{d_r}{d_l + d_r} p_r} \tag{19}$$

Where d_l is the distance between the pixels in overlap region to the border of the left matching image, and d_r is the distance between the pixels in overlap to the border of the right matching image.

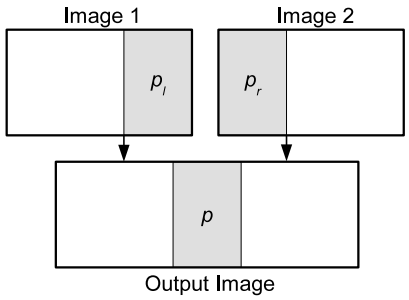


Figure 7: Blending in the overlap region

Chapter II System Realization

4 Design of Panorama Capture System

The system consists of three main modules – motorized head, control unit and battery pack. Head is based on Bescor MP-101 motorized pan/tilt head [15]. However, the head does not allow advanced control and there is no position feedback. These facts led to need of system modification. Mechanical parts with two DC motors were preserved, but all electronics was completely redesigned.

Head is controlled by Atmel Atmega328p microcontroller [16] [17] built on the prototyping board Arduino Pro Mini [18] with Push-Pull four channel driver L293B [19] that controls DC motors. Head also contains two voltage regulating circuits. The first serves as a power supply for the motors and the second for powering logical circuits. There is system for precise position detection based on diffusion reflection optocoupler and code circle for each direction.

Head is driven by control unit over I²C bus [20]. The unit is based on the Atmel ATmega2560 microcontroller [21] [22] placed on Arduino Mega 2560 [23] development board. USB host shield [24] is used for camera control and Bluetooth module [25] for communication with mobile devices. The unit also allows to connect Nintendo Wii Nunchuck [26] joystick for realtime position control and OLED[8] display for device status information.

The system is powered by external battery pack or other power supply. 9 – 12V DC (unregulated) is required for optimal functionality. The system is designed to maximal voltage of about 30 V but if higher voltage than 12 V is applied, voltage regulator can heat and can lead to gradual damage of components. The system overview is shown on image Figure 8.

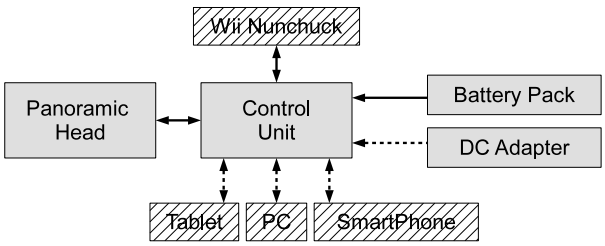


Figure 8: System overview

[8] OLED - Organic Light-Emitting Diode

5 Mechanical Construction

5.1 Panoramic Head

Mechanical construction of the head is based on commercial motorized pan/tilt head Bescor MP-101 [15]. Motion is ensured by pair of DC motors. Each motor is equipped by two worm gear in a cascade to achieve slow motion and large strength. Worm gear is fixed by the spring to minimize gear

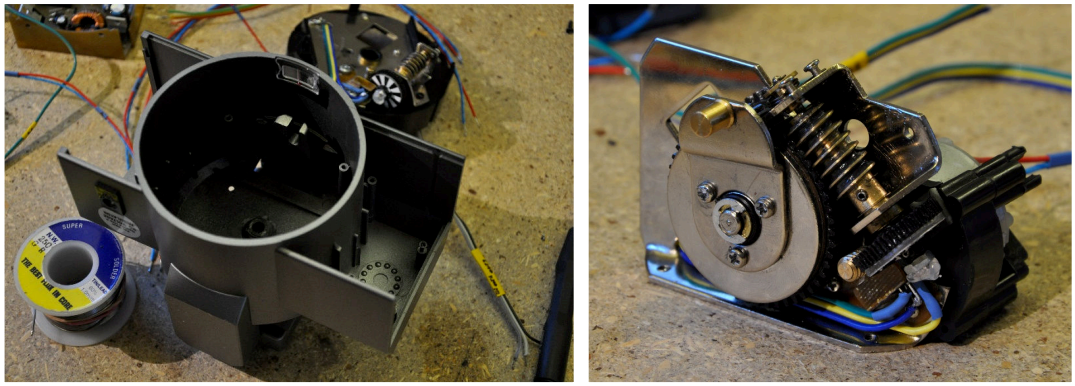


Figure 10: Left: Plastic box of the head without electronics, motors and gears, Right: Double worm gear with DC motor

clearance. Both gears are in ratio 1:50 that means the total ratio is 1:2500 and ensures very fluent a precise motion. This ratio is important for position detection because there are code circles placed on the second worm. Inner construction and all movable components are made of metal, the cover box and other parts are made of plastics. The box and the gear are on the Figure 10. Completed redesigned head is on Figure 9.

Figure 9: Panoramic head

Vertical and horizontal motion systems are completely separated. The design of the head and placement of components is shown on Figure 11. The front side of the head was also completely redesigned. All switches and indicators were replaced with new panel made of fiberglass and black painted.

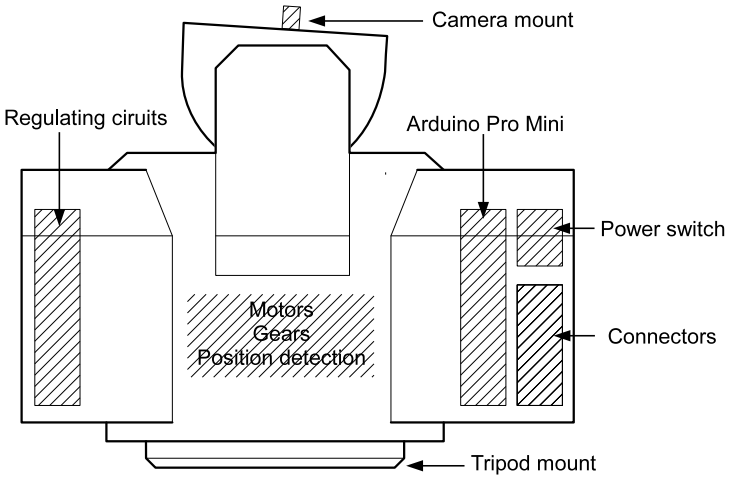


Figure 11: Side view of the head with schematic placement of components

All control features are realized by the control unit so all we need is the power switch, indicating LED, programming connector (for loading new firmware to the microprocessor) and communication connector (connection with control unit). Front view of the head is shown on Figure 12.

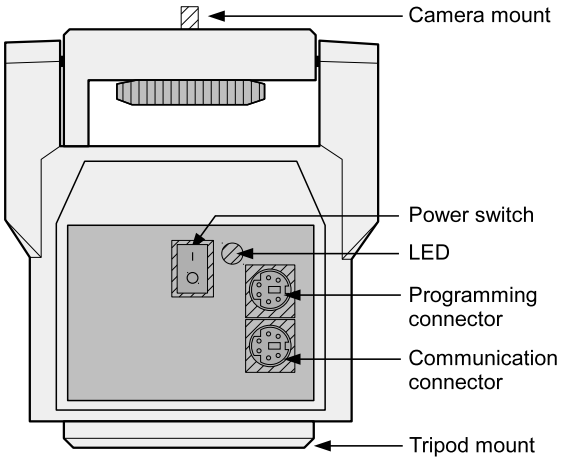


Figure 12: Front view of the head with the power switch, indicating LED and two DIN

5.1.1 Position Detection

There were several different possible principles of position detection considered. One of them was Gray code circle with photocell. This would allow to know absolute position without any initial calibration but there is problem with lack of space inside the head for this circle. In addition, the circle would have to be quite complex and it would have to have nine or ten bits to reach required precision. Finally, simple code circle with black and white stripes was selected. It is placed on the second worm gear and has 20 stripes. With the ratio 1:50 the resolution is calculated as $res = 20 \cdot 50 = 1000\ steps/360°$ or it can be expressed as 0.36°/step. Electronic system is able to move accurate to 1 step so the system works with the accuracy to 0.36°. The circle is illustrated on Figure 14. Optocoupler with the circle is on the Figure 13.

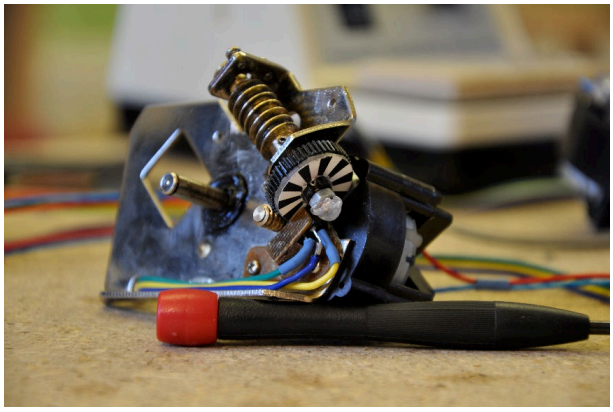


Figure 13: Position detection optocoupler with code circle

It was important to choose the right number of stripes so generating script in Processing language [27] was written. It allows to parameterize inner and outer diameter, colors and number of stripes. The result is automatically exported to PDF and printed to standard 80 g/m². The circle was sticked to polycarbonate disc and cut out to desired diameter. Special clean CD[9] without metal was used. In the end the code circle was painted by transparent oil varnish to protect the paper against potential condensation of humidity and to increase contrast. The stripes code circle was chosen because we want to achieve high contrast and maximal level difference at the detector output. 20 stripes was chosen as a value that provides enough accuracy and there is sufficient level difference on the output of the detection phototransistor.

Figure 14: 20 steps code circle

5.2 Control Unit

Control unit was designed for high durability and wide operational conditions. Dimensions and weight was also considered. Dural construction box was selected to withstand a fall from a height [28]. The box has plastic ends because of Bluetooth module usage. If the box was completely made of aluminum there would be problems with radio wave propagation so the combination of plastic and aluminum is ideal solution. The main part of the unit is Arduino Mega 2560 [23] with USB Host shield

[9] CD – Compact Disc

[24] that occupies almost the entire surface. The shield is connected directly to the Arduino so no wires or another connections are needed. There is small PCB[10] near by the Arduino board which powers I²C[11] busses, voltage conversion for Bluetooth module and amplification for the piezo buzzer. The buzzer is placed on the opposite side of the unit and it is mounted on a small PCB. The acoustic waves propagate outside box by the connectors and by the leaks around them so good audibility is guaranteed. Top view of the box and components are shown on the Figure 15.

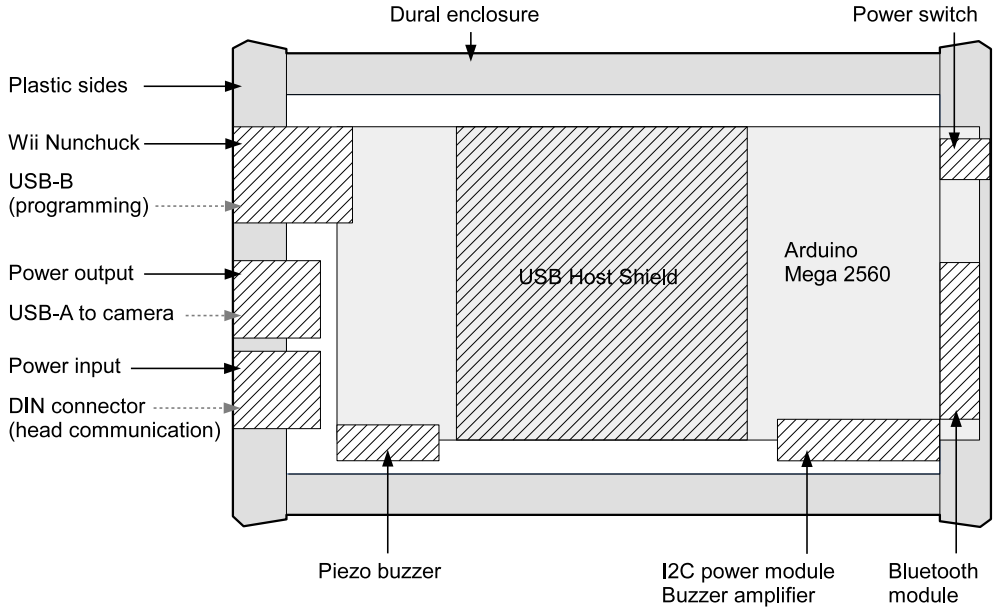


Figure 15: Top view of the control unit. The main part is Arduino Mega 2560 that occupies almost the entire surface of the unit. There are six connectors placed at the bottom side of the device (left side of the image) and power switch is at the upper side with the Bluetooth module (right side of the image).

There are six connectors on the bottom plastic end. The placement of the connectors is shown on Figure 17. There is a couple of equal power connectors (labeled as Power A and Power B) that are directly connected. One is used for power source input and the second one can be used for powering additional devices as lamp, smartphone charger or the camera. Both are low-voltage connectors with outer diameter of 5.5 mm and inner diameter of 2.1 mm. Connection between head and control unit is realized by six pin Mini-DIN connector. Only four pins are used for our application but there are extra two pins for possible purposes in the future. The unit communicates with the camera over USB[12] 2.0 bus [29] and uses USB-A connector. USB-B connector provides programming of the microprocessor, loading of new firmware, debugging and configuring of the unit. Wii Nunchuck connector [26] is used to control by the joystick.

[10] PCB - Printed Circuit Board
[11] I²C - Inter-Integrated Circuit
[12] USB – Universal Serial Bus

On the top plastic side there is a power switch and Bluetooth module. The module was placed as close as possible to the side and it is fixed by double-sided high adhesive tape. Both sides are mounted by four screw. Final version of the control unit is on Figure 16.

Figure 16: Control unit

On the top dural side there is 1.96" OLED display which displays important information about the status of the head and the control unit. The display is covered by double layer glass. Special precise glass was used for this purpose. Both layers were cut to appropriate shapes, glued by transparent glue and mounted to the aluminum case.

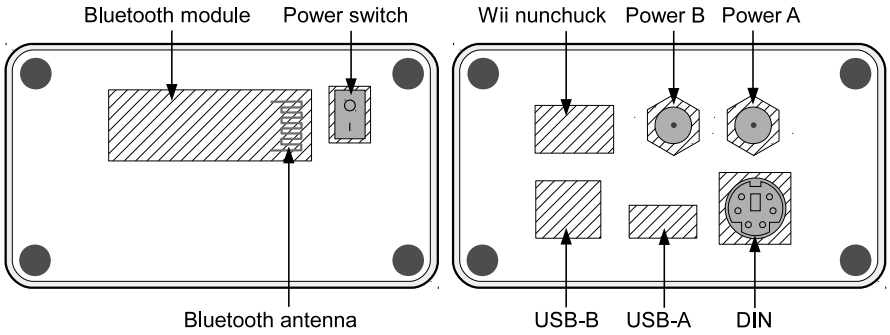


Figure 17: Side view of the control unit

5.3 Dimensions and Weights

All the system was designed to be as small and light as possible, but mechanical construction of the head requires some weight. The head weights about 810 grams with dimensions of 15×10×7 cm. The control unit weights about 280 grams with dimensions of 13×8×4.5 cm. The battery pack can be replaced with arbitrary battery pack or other power supply. The default battery box has dimensions about 13×6.5×3.5 cm with the weight of 280 grams. The whole system weighs approximately 1370 grams.

6 Electronics

6.1 System Architecture

Electronic system is divided to two main units. Panoramic head and control unit. The control unit includes Arduino Mega 2560 board [23] which is the brain of the entire system. There are three main modules inside the unit. Bluetooth module JY-MCU HC-06 provides wireless communication with smartphones, tablets, laptops or other mobile devices. USB host shield communicates with Arduino and enables to plug the camera in with USB interface. OLED display shows basic information about system activities and statuses. It can also display error alerts or completed job information. The unit is programmed, configured and controlled over USB 2.0 port. This port serves also for debugging purposes. The unit also allows the connection of Nintendo Wii Nunchuck joystick for real-time position control. Power supply is provided by external battery pack or adapter.

The brain of the head is the microcontroller Atmel AtMega328p placed on prototyping board Arduino Pro Mini [18]. It controls all motion processing and communication with control unit. Head

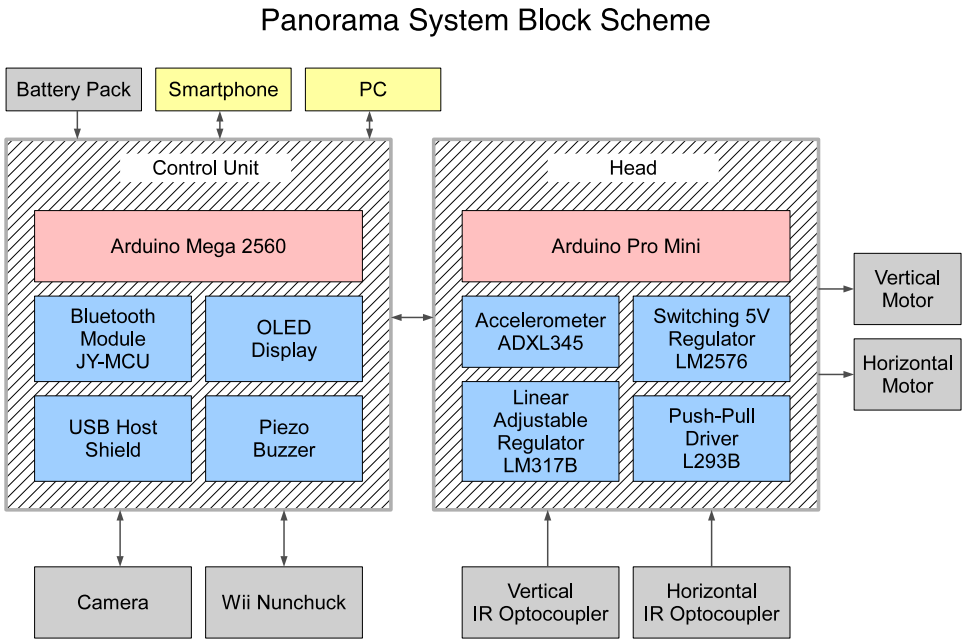


Figure 18: Block scheme of the panorama system

is powered by unregulated voltage source which is led to two voltage regulators. The first regulator is linear for powering logics, Arduino, motor driver and optocouplers. The second one is adjustable switching regulator used just for powering the motors. Motors are driven by integrated circuit Push-Pull driver which accepts logical signals from Arduino and switch the adjusted voltage to appropriate motor and direction. The head is equipped by accelerometer that can be used for detection of shaking or unwanted movement of the head and send reports to control unit. The scheme is illustrated on Figure 18.

6.2 System Communication

Communication model is illustrated in Figure 19. Most of components communicate over I²C bus, USB Host shield communicates over SPI and BT module and PCs over serial bus. Also the most important connection between control unit and the head is realized just by I²C bus.

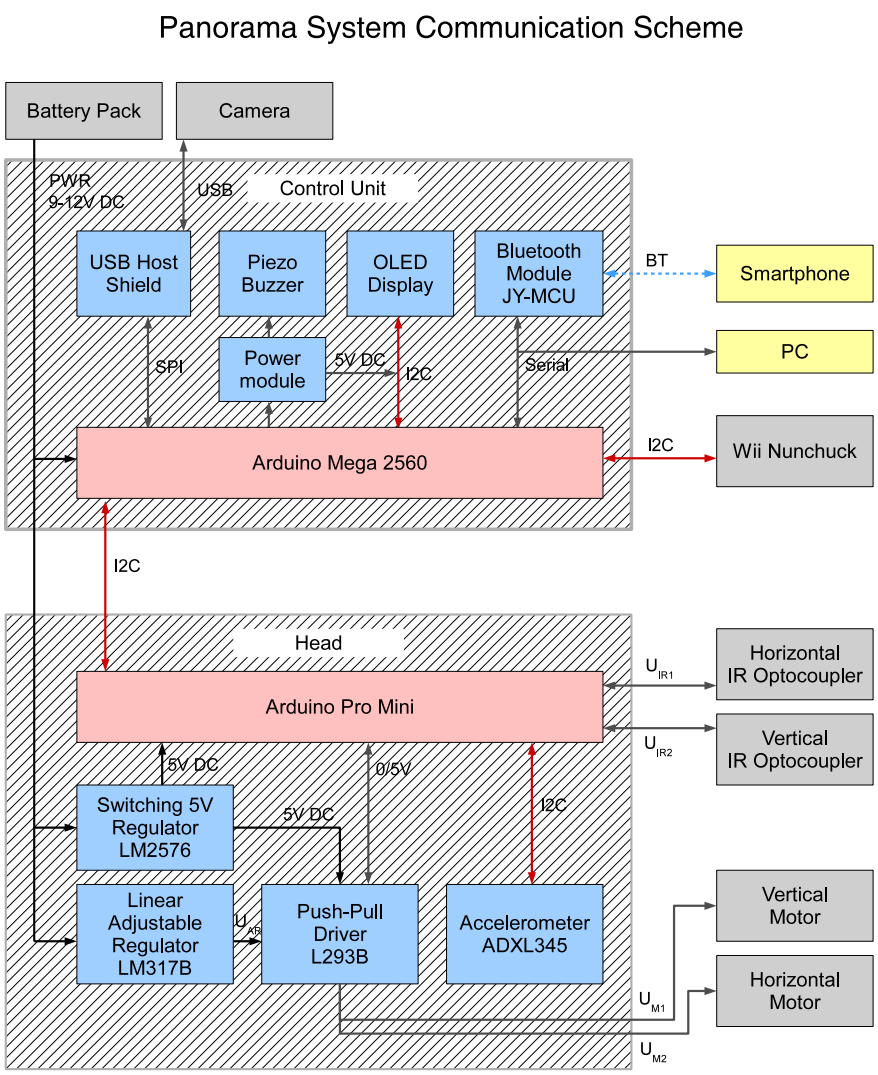


Figure 19: Communication scheme of the system

6.2.1 Buses

A. I²C (Inter-Integrated Circuit)

I²C is multi-master, multi-slave serial computer bus [20]. It uses two bidirectional open-drain lines. Serial Clock Line (SCL) and Serial Data Line (SDA). Both lines are connected by the pull-up resistor to positive voltage (5 or 3.3V) which keep high level in the idle state. It enables to connect up to 128 devices (with 7 bit address scheme). Maximal frequency is specified to 100 kHz or 400 kHz according to the version. The bus allows to connect many types of devices as EEPROMs[13], displays, A/D[14] or

[13] EEPROM - Electrically Erasable Programmable Read-Only Memory
[14] A/D – Analog to Digital converter

D/A[15] convertors, microcontrollers and other modules. The schematics of the bus is illustrated in Figure 20.

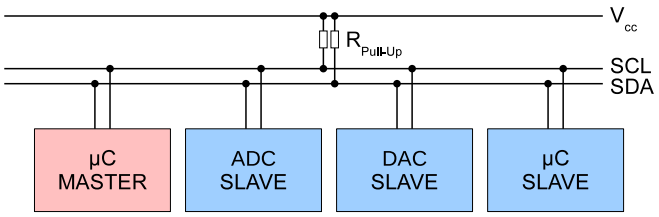


Figure 20: Schema of I2C bus

B. SPI (Serial Peripheral Interface)

SPI is serial bus designed for communication among microprocessors and other integrated circuits. It also allows to connect EEPROMs, A/D converters, displays etc. [30] Addressing is solved by external lines, which activates selected device when logic zero is on the line. SPI is single-master bus. That means that there is just one master device that controls the bus clock and the communication. There are four lines: SCLK (Serial Clock), data lines MOSI (Master Out Slave In) and MISO (Master In Slave Out) and SS (Slave Select) lines for addressing. Typical frequency ranges up to a few MHz. SPI bus is illustrated on Figure 21.

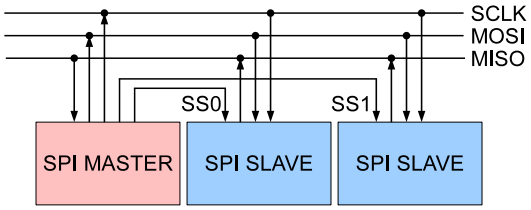


Figure 21: Schema of SPI bus. There are two data lines (MOSI, MISO), serial clock line and slave select lines

6.2.2 Head Communication Interface Specification

The head disposes two six-pin Mini DIN connectors. One for communication between the head and control unit and the second one for head programming. Connector pins are shown on Figure 22.

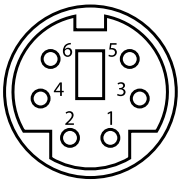


Figure 22: Pins on Mini-DIN connector

[15] D/A – Digital to Analog converter

A. Head – Control Unit Communication

Only four pins are used for head to control unit communication. The first couple is DC power (Ground and VC+) that leads voltage in range of 9 to 12V DC. The second couple is I²C bus (SCL and SDA). Positions of wires on the connector are in Table 1.

Pin number	Wire description
1	GND (Ground)
2	SCL (I²C clock)
3	VCC (Supply voltage)
4	SDA (I²C serial data)
5	Not connected
6	Not connected

Table 1: Pin configuration of Mini-DIN connector (communication between head and control unit)

B. Head Programmable Interface

Programmable interface is realized over serial bus, which in connected to serial-to-USB converter and to PC. The bus utilizes two data lines RX (Serial Receive), TX (Serial Transmit), VCC (Power supply), ground and reset line. Position of the pins of Mini-DIN connector is shown on the Table 2.

Pin number	Wire description
1	RXI (Serial Receive In)
2	VCC (Supply voltage)
3	TXO (Serial Transmit Out)
4	GND (Ground)
5	DTR (Reset)
6	GND (Ground)

Table 2: Pin configuration of Mini-DIN connector (head programming)

6.2.3 Communication Management

Communication management is divided between control unit (which controls main buses) and head that ensures motion of the head.

A. Control Unit

Arduino inside control unit is the master device for the I²C bus and controls the communication. It sends commands to display and receives signals from Wii Nunchuck. It can also read signals from accelerometer in the head. The main task is the communication with the head. The set of commands is used for the communication between both Arduinos. Details will be explained in the Software chapter. USB Host Shield is fixed directly on Arduino and connected with SPI bus. The shield serve as a module where camera USB cable is plugged.

B. Head

The head is responsible for camera motion. Arduino receives commands from control unit over I²C bus and sends TTL[16] signals to Push-Pull motor driver L295B. Voltage levels are 0 or 5V DC. Motor voltage is in the range from 3 to 5V DC according to regulator configuration but it is the same

[16] TTL - Transistor–Transistor Logic

for both the motors and cannot be changed by software [19]. For adjusting of the speed of the motion, there is a potentiometer on the board. Driver switches the voltage set in adjustable linear regulator to both motors according received signal. It can also switch motion direction by change of polarity.

Second very important module is pair of ID (Infra-Red) optocouplers that measure the exact position in horizontal and vertical direction. For output signal, the intensity of IR diode is important. Therefore there are two potentiometers on the board that can adjust IR diode voltage and its intensity. Each optocoupler can be regulated separately.

The last module in the head is the accelerometer. It is connected over I²C bus with all the system and its output can be read from both Arduinos. It provides detection and warning if some unwanted movement or shaking appears. The electronics of the head is shown in Figure 23.

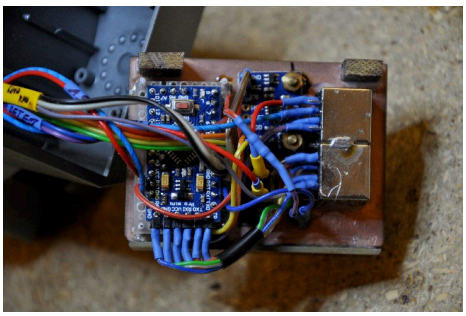


Figure 23: Arduino Pro Mini and Accelerometer in the head

6.3 Electronic Design

The circuit consists of a few blocks which will be described separately. The schematic was created in software Eagle and it is shown in Figure 24. The head and control unit work as separate modules. Powering of both modules is also designed as independent and both have own input voltage regulating circuits. Control unit is powered by 9-12 V DC from battery pack or power supply and the same voltage is applied to head over DIN connector. There is power switch at the supply input. Operating voltage of most components is five volts, only the motors are powered by lower voltage. The communication between both units is provided just by I²C bus. The bus is powered in the control unit by five voltage Arduino's output with two pull-up resistors. It means that if the head was used with another control unit, it would also have to provide I²C bus powering.

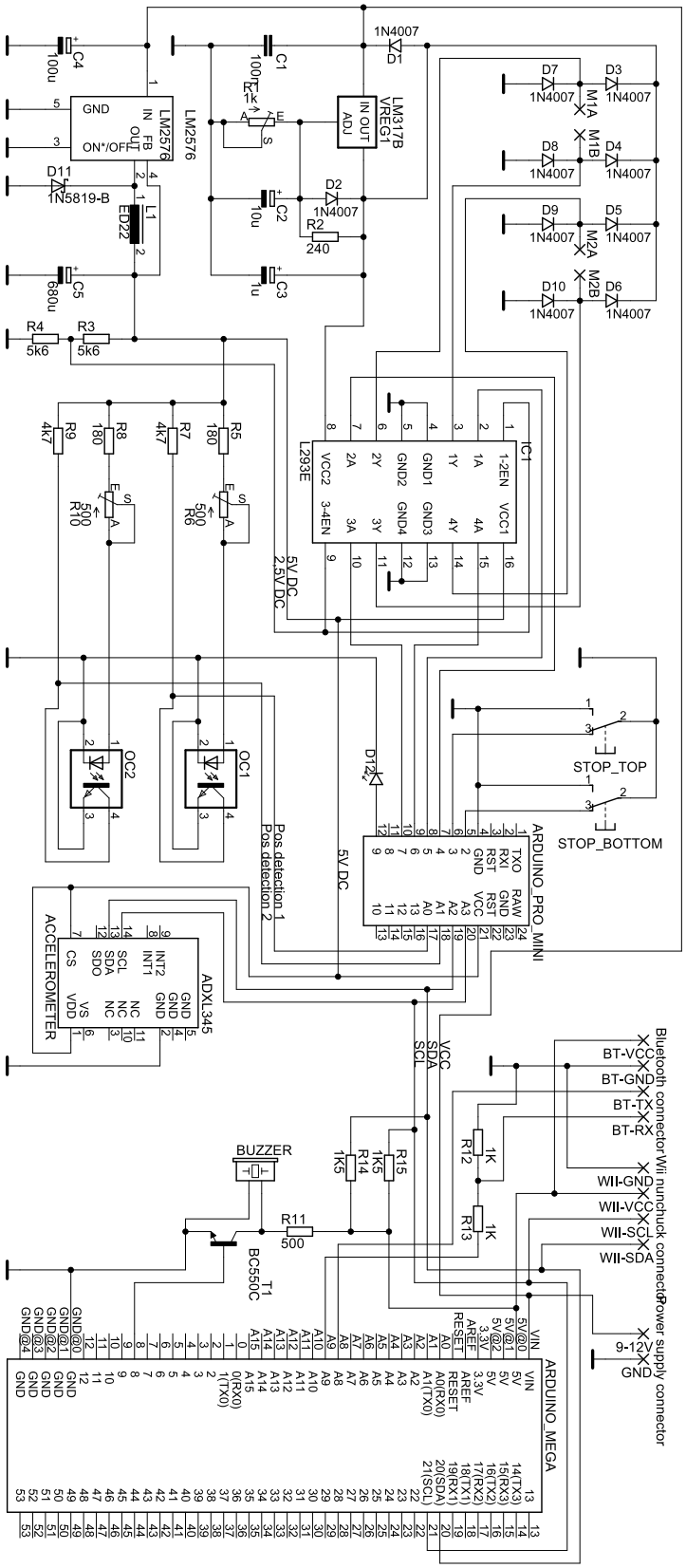


Figure 24: Panorama system schematic circuit

6.3.1 Head

The main blocks of the head are linear and switching voltage regulators, Push-Pull motor driver with protective diodes, position detection system with optocouplers, protection switches in end positions of vertical movement and Arduino prototyping board.

A. Arduino Pro Mini

Arduino is open-source prototyping platform based on Atmel microcontroller [18]. Pro Mini is on the smallest version of the Arduino product line. It is based on ATmega328 microcontroller. It has 14 digital input/output pins, 6 analog input pins with 10-bits A/D converters, on-board resonator and six pin header. There are two versions of the Pro Mini. One runs on 3.3 V and 8 MHz, the other at 5 V and 16 MHz. The 5 V variant was chosen for our purpose because of higher frequency and common 5 V, which is used by many other modules in the head. It has 32 kB flash memory and 2kB SRAM[17]. The capacity of the RAM is not so high, but it is enough for our application and for all simple software for which is Arduino intended. There are 6 pins that provide 8-bit PWM[18], SPI, Serial and I²C interface.

For use of these interface there are many libraries that implements appropriate protocols and provide simple user interface. The libraries are written in C++ language and included to the Arduino sketch.

B. Motors

There are two linear motors for two motion direction. Motors are driven by Push-Pull driver L293B [19] which has 4 digital signal control inputs, 4 power outputs, Vss – logic supply voltage (5V DC) and Vs – supply voltage. Vss powers the logic circuit and Vs is switched to driver outputs according to input signals statuses and their combination. Input signal takes states of logic zero (0V) or logic

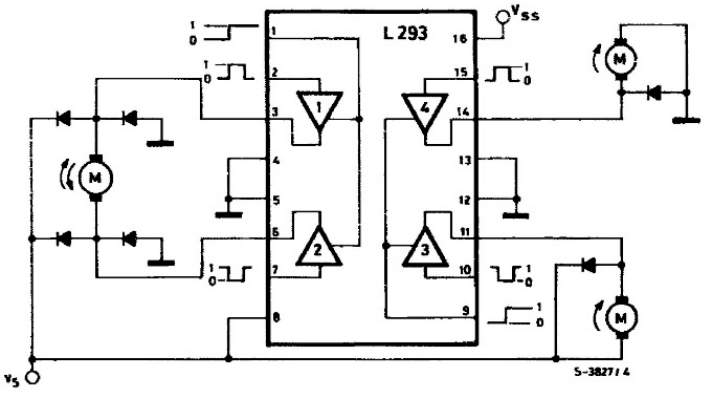


Figure 25: L293B Push-Pull driver functional diagram [19]

Vi (each channel)	Vo	Vinh (Chip Select)
H	H	H
L	L	H
H	X (*)	L
L	X (*)	L

Table 3: L293B Push-Pull driver truth table [19]

[17] SRAM - Static Random-Access Memory
[18] PWM – Pulse-Width Modulation

one (5V). The driver schematic diagram is on the Figure 25. There is also CS (Chip Select) pin (also called V_{inh}) which enables outputs. Relationships between input signal and CS pin are in Table 3. V_s is output voltage from adjustable linear voltage regulator.

C. Linear Voltage Regulator

This adjustable regulator is used to set motor supply voltage. This output voltage influences the speed of motion of the head. The exact value of output voltage was set experimentally and it ranges from 3.5 V to 5V. Finally the voltage was set to approximately 4.2 V. The circuit is sketched on the Figure 24 and is based on the common LM317B chip which provides an internal reference voltage of 1.25 V between output and adjustment terminals. This is used to set a constant flow across an external resistor divider. The output voltage is given:

$$\boldsymbol{V_O = V_{REF}\left(1 + \frac{R_1}{R_2}\right) + I_{ADJ}R_2} \tag{20}$$

The device is designed to minimize I_{ADJ} max 100 μA and V_{REF} is 1.25 V. If we want output voltage from 3.5 to 5 V the 1 kΩ potentiometer was selected. Diode D1 protects the device against input short circuit and diode D2 protects against output short circuit for capacitance discharging. The capacitor C3 improves the ripple rejection of about 15 dB and output tantalum capacitor C2 improves transient response. C1 capacitor is placed for input voltage filtering [31].

D. Switching Voltage Regulator

Switching voltage regulator is used to supply logic modules and other circuits. It is based on integrated circuit LM2576 with output voltage of 5 V. The circuit is sketched on the Figure 24. The inductance of the coil L1 was determined to 470 μH according to the table in LM2576 datasheet when maximum input load was 0.5A. The coil must be able to work on frequency of 52 kHz. Output capacitor C5 was determined by the datasheet to 680 μF and input capacitor C4 to 100 μF for stable operation. The Schottky diode D11 protects the chip against short circuit [32].

E. Position Detection

Position detection is realized by two optocouplers that consists of IR LED on the wavelength of 960 nm and phototransistor in common-emitter variant [33]. The intensity of the LEDs is regulated by the adjustable resistors R6 and R10. The intensity has big influence to output signal and it can affect the signal offset. The output from phototransistor is connected to Arduino analog input pins A0 and A1.

Output signal is illustrated on Figure 26. It was analyzed by improvised oscilloscope based on another Arduino unit [34].

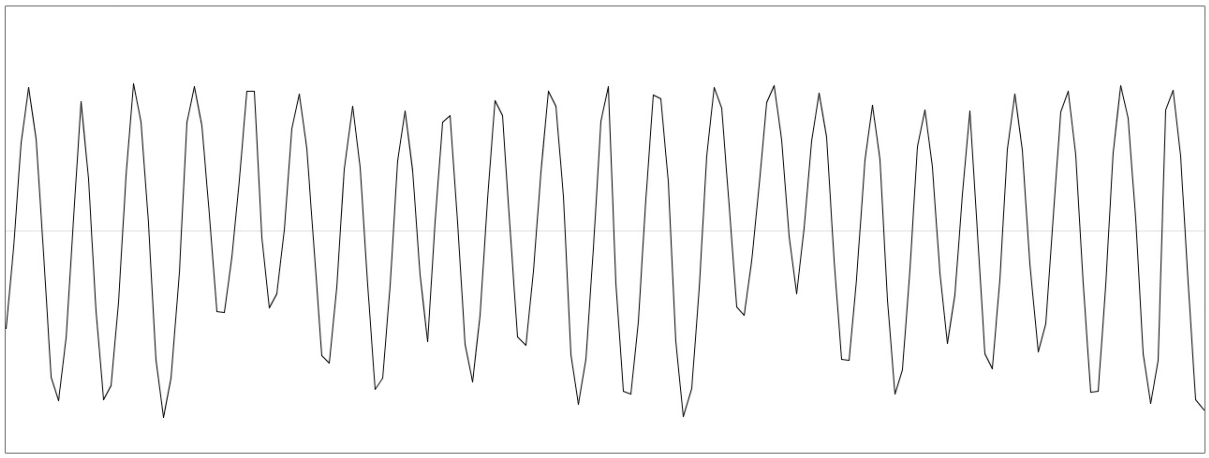


Figure 26: Output signal from horizontal position detection phototransistor. Full displayed range is 5 V

6.3.2 Control Unit

Main part of the unit is Arduino Mega 2560 prototyping board. It is based on microcontroller Atmel ATmega 2560 and it offers more features than Arduino Pro Mini used in the head. The frequency is the same (16 MHz) but important feature is 256 kB Flash memory (that is much higher value and practically no code limitation). As next, there is 8 kB RAM. It has also 54 digital input/output pins and 16 analog input pins. Arduino Mega also enables to connect USB Host shield which provide USB 2.0 connector for camera connection. The core of the shield is USB controller MAX3421E which communicates with the Arduino board using SPI bus (through the ICSP[19] header) on digital pins 10, 50, 51, and 52. These pins cannot be used for other purposes. The shield provides 500 mA to a connected device.

I²C is connected to pins 20 (SDA) and 21 (SCL). Both lines are pulled up with 1500 Ω (R14, R15) resistors to regulated 5 V at the Arduino output.

Bluetooth module communicates over serial interface and is connected to pin A8 (RX[20]) and A9 (TX[21]). There is small complication with Bluetooth module because its operation voltage is 3.3 V (logic signal) and Arduinos operational voltage is 5 V. Therefore, voltage divider is used to convert Arduino transmit 5 V signal to Bluetooth 3.3 V signal. The divider consist of two resistor with the ratio of 1:2 (R13 – 1kΩ, R12 – 2kΩ). Arduino RX pin and Bluetooth TX pins are connected directly because Arduino accepts 3.3 voltage level.

Wii nunchuck connector is connected to I²C bus without any modification and it can be eventually used for another devices using this bus.

Last module is piezo buzzer (LD-BZPN-1705) that beeps when some important event occurred. It has resonant frequency of 4±0.5 kHz, nominal current of 1 mA and nominal voltage of 5 V. The sound intensity can reach up to 78 dB (in the distance of 10 cm). The signal is generated on the pin 8 and amplified by low noise transistor T1 (BC550C). The output is connected in common emitter variant with collector resistor R11 (500 Ω).

[19] ICSP - In Circuit Serial Programming
[20] RX – Serial Receive
[21] TX – Serial Transmit

6.4 PCB Design and Technology Processes

The enclosure of the head is quite small and there is lack of space for electronics. PCB was designed for the circuits inside it. The board contains practically all electronic components in the head except Arduino module and accelerometer. The board was designed with software Eagle and the layout is shown in the Figure 28 and the result in Figure 27.

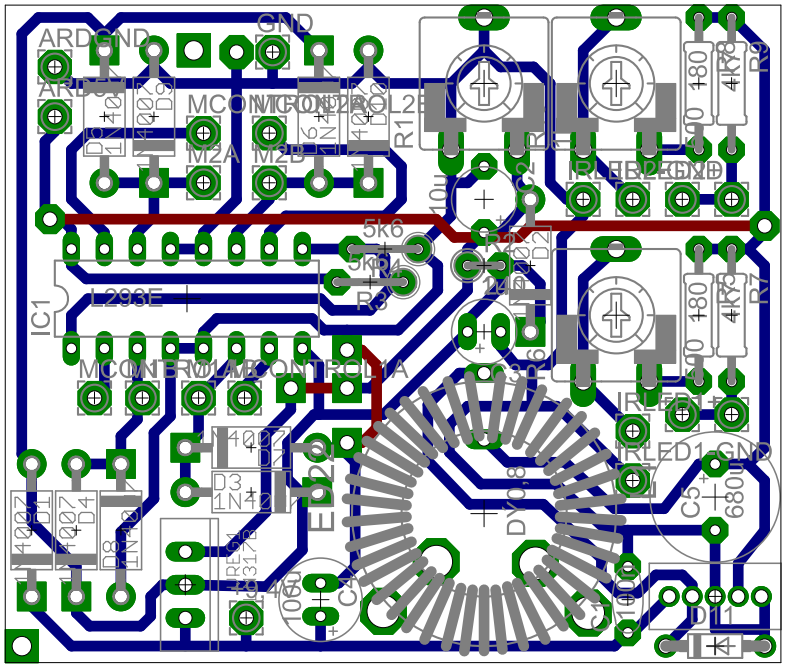


Figure 28: layout of PCB for the head

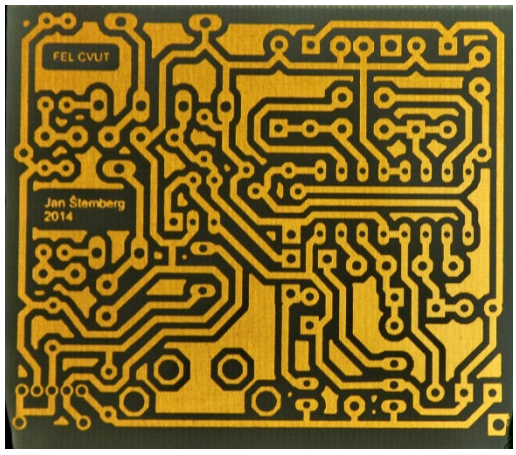


Figure 27: Finished PCB

The technology process is divided to several steps:

1. Printing of the PCB to transparent foil by laser printer with the resolution of 300 DPI
2. Pattern smoothing – printed pattern contains some inhomogeneities in black areas because of uneven deposition of tonner. These places caused problems in etching process. For smoothing of the pattern the foil is attached to glass and moved over the bowl with warmed acetone. The warming of the acetone must be very slow and its temperature must not reach over about 40°C

otherwise there is a risk of explosion. The smoothing process takes about 10-20 second. It is important to take the pattern down otherwise the details will be lost. After the process the foil must let dry on the air.

3. Desired shape of the PCB is cut out of photocuprextit. We can use finished photocuprextit or normal cuprextit covered with a ply of positive photoresist.
4. UV lamp is used to enlighten the pattern from the distance of about 12 cm. The foil is placed between two clean glass plates. The time of lighting is very important and it was experimentally determined to 8 minutes.
5. Enlightened patter is dipped to developer (1.5 % sodium hydroxide) for 5-6 minutes. It is good practice to move with the brush over the board for uniform and faster process. The board is washed with water after the process.
6. The etching solution for PCB (Ferric Chloride) is heated to about 50°. The warmer solution the faster process. The board is dipped to etching solution for about 8 minutes. The same trick with the brush is helpful. It is important to monitor the thickness of copper layer and adjust time of the process.
7. If the board is not soldered in a short time after etching, it is good not to remove rest of photoresist because it protects the copper against oxidation. The second way to protect the board is to remove the photoresist and use soldering varnish that allow soldering for more than one year without risk of copper oxidation.

7 Software Implementation

Software for the microcontrollers is developed in Arduino language which is practically C++ language with some additional functions added. It uses also C++ compiler so there is no significant efficiency loss [35]. This solution has big advantage of simple usage of many functions, libraries that are otherwise complex problems (if the software is developed in clean C or C++ language). The language also allows direct access to all registers of Atmel microcontrollers so all the parameters can be set manually. The Arduino platform is very popular and there are many users of these boards. That means that there is a lot of manuals and tutorials for many tasks and problems.

Each program consists of several blocks of code. The code starts with `#define` and `#include` statements which enable to include additional libraries and define constants. These statements are followed by two important functions. The `setup()` and `loop()` function. The `setup()` function is run only once and is called when program starts. It is used to initialize variables, start using libraries or set pin modes. The `loop()` function loops consecutively over again. Both functions are type of void that means that they does not return any value.

The basic structure of the code is as follows:

```
#include <MemoryFree.h>       // include additional libraries
int buttonPin = 3;            // initialize global variable buttonPin

void setup()
{
  // code that will be run only ones when Arduino starts
  Serial.begin(9600);         // start serial communication
  pinMode(buttonPin, INPUT);  // set pin 3 as input
}

void loop()
{
  // code that will be run over again
}
```

Because of complexity of the code, only block diagrams and some code samples will be presented. Complete codes are included as appendix. The chapter is divided into main three parts. There are some principles which are used both in the head and control unit software. Next two parts describe important behavior of head and control unit software.

7.1 Common Principles

7.1.1 Using Flags

Programming of the microprocessors is very "low-level" which requires some specifics. One of the most important things is processor time. If we had some time-consuming operation, for example loop with many repetitions and computationally complex content, it would block all other operations. This may also occur when just `delay()` command is used. It accepts number of milliseconds of waiting as an argument. The implementation of this function can be explained as previously mentioned loop that repeats until difference of time from the loop start to actual time is lesser than given argument. Also in this case, all other activities are blocked. This can be critical when some software pin interrupt occurs or when some bytes come to Arduinos buffer and need to be served before buffer overflows. The solution of this situation is using flags. If we do some time critical operation that requires longer time to run, we will store all parameters needed for the task and set a flag that data are ready for computing. Parameters are usually stored as a global variables but we can also use structures or objects,

because Arduino accepts almost all features of C++ language. We just have to keep in mind that the preprocessor works differently than in C++. The consequence of this behavior is that we have to declare all own data types in header files (.h files) and include as an external library with help of `#include` statement. The flags themselves are almost always stored in global variables because there is no need for a lot of data. Usually boolean type of variables are used because the logical information (`TRUE` | `FALSE`) is just what we need. The flag is set by this way and when there is no time critical operation, flag is detected in `loop()` function and the operation is runs.

7.1.2 Serial Communication

Serial communication is handled by Serial library which is built in basic Arduino library and it may not be additionally included. The serial communication must be initialized by `begin()` method in `setup()` function. The parameter is baud rate which gives the speed of data transfer. Default rate is 9600 bauds and PC accept values up to 115200 bauds. Initialization look like this:

```
void setup() {
  Serial.begin(115200);
  …
}
```

Serial receive pin (RX) provides hardware interrupt which call function `serialEvent()`. It means that when some data come to the buffer, this function is called to process the data. There is another function `available()` which returns `TRUE` if there are bytes in buffer to read. It is used as a condition in while loop which repeats until there are some data to read. The data is read by function `read()`. It reads first byte and shifts the buffer. The read() returns type of byte (unsigned 8-bit number) which can be converted to char type if we use ASCII[22] characters in communication. ASCII is also represented by 8-bit number. The characters are read byte after byte and stored into global variable `inputSerialString`. When control character is received, the flag `inputSerialStringComplete` is set to `TRUE`. Control character indicates the end of the serial command. There are three control characters: `\n` – new line character, `\r` – carriage return and `$`. All of them may be used to terminate the command. The code for serial receive may look like this:

```
void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputSerialString += inChar;
    // if the incoming character is a newline or a carriage return, set a flag
    if (inChar == '\n' || inChar == '\r' || inChar == '$') {
      inputSerialStringComplete = true;
    }
  }
}
```

Flag `inputSerialStringComplete` is tested in the `loop()` function and if `TRUE` is detected then `inputSerialString` can be parsed and processed. Flag is set back to `FALSE` at the start of processing function.

Serial transmit is realized by `print()` or `println()` function which accept string argument and output is sent as sequence of bytes. There is also `write()` function which accept one byte or char (8-bit integer) and sends it as binary input. There is a small sample of sending data over serial:

```
Serial.println("Send data by serial");
```

[22] ASCII - American Standard Code for Information Interchange

In addition to default serial port, we can use so-called software serial communication. It can be attached to any port that support change interrupt. This feature is useful when more than one serial port is needed. This is used in control unit. Standard serial port is used for communication with PC for debugging purposes and software serial for communication with Bluetooth module.

7.1.3 I²C Communication

Communication over I²C is provided by Wire library [36] that must be included before `setup()` function is declared.

```
#include <Wire.h>
```

I²C library is initialized by function `begin(int address)` which accept argument with the address of the device itself. Then we must register callback to set the function that will be called when data are received.

```
void setup() {
  …
  Wire.begin(4);                // join i2c bus with address #4
  Wire.onReceive(I2CEvent);     // register event
  …
}
```

Loading there is similar to reading serial input. There is a small difference in the storing – commands sent over I²C are four bytes (this is defined by the author). First two bytes are stored separately and third and fourth bytes gives one 16-bit integer. MSB[23] is sent as first. Variable `I2Ccommand` is defined as array of integers.

```
void I2CEvent(int howMany)
{
  byte commandByte[4];
  int i = 0;
  while(0 < Wire.available()) {
    commandByte[i] = Wire.read();      // receive byte
    i++;
  }
  I2Ccommand[0] = commandByte[0];
  I2Ccommand[1] = commandByte[1];
  I2Ccommand[2] = commandByte[2];      // store MSB
  I2Ccommand[2] = I2Ccommand[2] << 8;  // bit shift by 8 bit
  I2Ccommand[2] |= commandByte[3];     // store LSB[24]
  I2CCommandCompleted = true;          // set flag
}
```

When command is loaded, flag is set and tested by the same way as serial library.

7.2 Head

The most important activity of the head software is position control. This activity is divided into two main parts. These are position detection and controlling the motors.

[23] MSB - Most Significant Byte
[24] LSB - Least Significant Byte

7.2.1 Position Detection

Position detection is realized over reflex optocouplers whose outputs are connected to pins A0 and A1. Typical output signal is shown in Figure 26.

To determine changes between white and black stripes we need to use interrupt. Interrupt is special process that checks input pin states and when some change occurs, it stops actual processor activity and run the block of code defined by the interrupt. There is a small complication that there are only two external hardware interrupts on Arduino Pro Mini and they are not equipped by 8-bit A/D converters. Therefore, we have to use special library `PinChangeInt.h` [37]. It enables software implementation of interrupt which can be attached to any pins on Arduino board. The software interrupt is a little bit slower compared to hardware interrupt but it is absolutely sufficient for our detection. For the initialization of this library we have to attach interrupts to desired pins. There is function `attachPinChangeInterrupt(pin, callback, mode)` which accepts three arguments. The first is the watched pin, second is a function callback which gives the name of function that will be run when interrupt occurs and last argument is mode that sets when the interrupts fires. The function accepts three modes: `RISING`, `FALLING` and `CHANGE`. We need to detect both transition so `CHANGE` mode is used. Setup for our interrupts looks like this:

```
void setup() {
  …
  attachPinChangeInterrupt(vDetectPin, verticalStep, CHANGE);
  attachPinChangeInterrupt(hDetectPin, horizontalStep, CHANGE);
  …
}
```

These lines tells that when interrupt on `vDetectPin` occurs, function `verticalStep` is called. The implementation of the function is very simple:

```
void verticalStep() {
  if(movingUp == 1) {
    vStepCount++;
  } else {
    vStepCount--;
  }
}
```

Variables `vStepCount` and `movingUp` are global variables. `vStepCount` counts steps from zero position. The interrupt cannot detect the direction of code circle rotation, so we have to use flag `movingUp` which is set by another function when motion starts. If the head is moving up the counter is incremented otherwise the counter is decremented. Variable `vStepCount` is declared as signed integer so it may acquire negative values.

The motion range in vertical direction is about ± 15° from zero position. Therefore, the head needs to be calibrated in the vertical direction when it is powered on. The calibration process consists of several steps. The head moves down until bottom end is reached and then it moves up by exactly defined number of steps. Horizontal position may not be calibrated because the head can rotate continuously around.

7.2.2 Motor Control

Motor control is realized by setting digital ones or zeroes to appropriate pins. Each horizontal and vertical direction is driven by two pins. Before we set logical value on the pin, we have to set the pin as an output. There is function `pinMode(ledPin, OUTPUT)` for this reason. The function accepts two arguments – first pin with whom we want to work and second the output mode (can be one of these: `OUTPUT`,

INPUT, INPUT_PULLUP). When pin mode is set, we can use digitalWrite(ledPin, value) function to set the value. The value can be HIGH or LOW. So if we want the head to move up, we set upPin to HIGH state and downPin to LOW state.

The process of position control is visualized on flowchart in Figure 30 and is explained for vertical direction. Horizontal movement is realized by the practically same way. It starts by calling moveToVerticalDeg(float posDeg) function which accepts argument posDeg. This is desired position in degrees. This argument is converted to number of steps on code circle by vDeg2Step(float posDeg) function and stored in vTarget variable. Actual position of the head is stored in global variable vStepCount. If vTarget is bigger than vStepCount, the head will move up otherwise down. moveUp (respectively moveDown) function is called and pins are set to appropriate states. Flag movingUp is set to 1 that indicates that the head is moving up.

Interrupts attached to detection pins are watching the signal and if CHANGE event occurs, verticalStep function is called. movingUp flag is tested and vStepCount is incremented receptively decremented (if movingUp is FALSE and interrupt is fired, it means that the flag moveDown is set to TRUE). This process proceeds until actual heads position vStepCount is bigger (resp. smaller) than the desired position vTarget. When this occurs vTarger is reset and feedback to the control unit is sent. After that stopUp or stopDown function is called.

There is a small problem resulting from mechanical construction. There is a big gear ratio between motor and the head construction. It is 1:2500 that means motor is practically not loaded and slowed with external forces and inertia of the motor causes that motor does about 2 stripes moreover on the code circle. Therefore small trick is used. When counter says that the head is on the right position, stopUp or stopDown function is called and they repolarize motor power supply for very short time (about 20-30 ms). This is realized by variable vReverseTime which is set to actual time plus 30 ms and flag vMovingReverse which is set to TRUE. In the loop function, this flag is tested and actual time in milliseconds obtained by millis()[25] function is compared to vReverseTime. If both these condition are evaluated as TRUE then both pins are set to LOW state flag vMovingReverse is set to default zero and process is completed.

This solution stops the motion almost immediately. This quite complex implementation of timing is necessary because this process must not block other time critical processes.

7.2.3 Detection of End Stop Positions

Vertical motion range is limited to about 60°. To protect internal mechanism there are sensors which detect if head reach up or down end stop position. The sensor is realized as a switch. One contact of the switch is grounded and the second connected to Arduino pin. The mode of the pin is set to INPUT_PULLUP which means that the pin is pulled up by internal resistor (usually 20-50 kΩ) to 5 V. So if the head is in central position range, the switch is disconnected and pin state is HIGH. If one of the end stops is reached, the switch closes, pin is grounded and pin state is LOW. These states are watched and if any stop is detected motion is stopped and event is sent to the control unit.

[25] millis() - function returns number of milliseconds since the Arduino was powered on

7.2.4 Code Analysis

The code has about 600 lines, size of binary sketch is about 16 kB (approx. 50 %) and program uses about 1100 bytes of SRAM (approx. 56 %). Interesting is the duration of the one loop of the `loop()` function. Its dependency on loop number is shown in Figure 29. This can be interpreted as a duration of processor cycle.

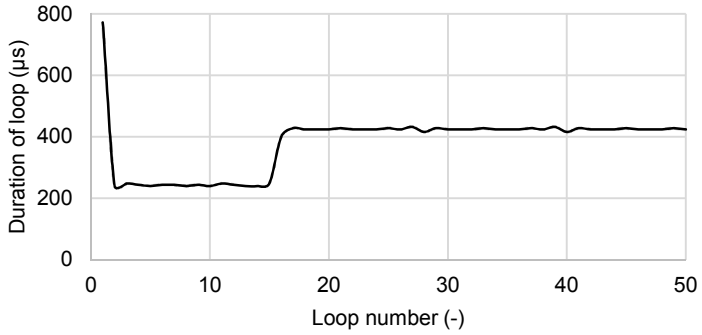


Figure 29: Dependency of the loop duration on loop number since Arduino starts

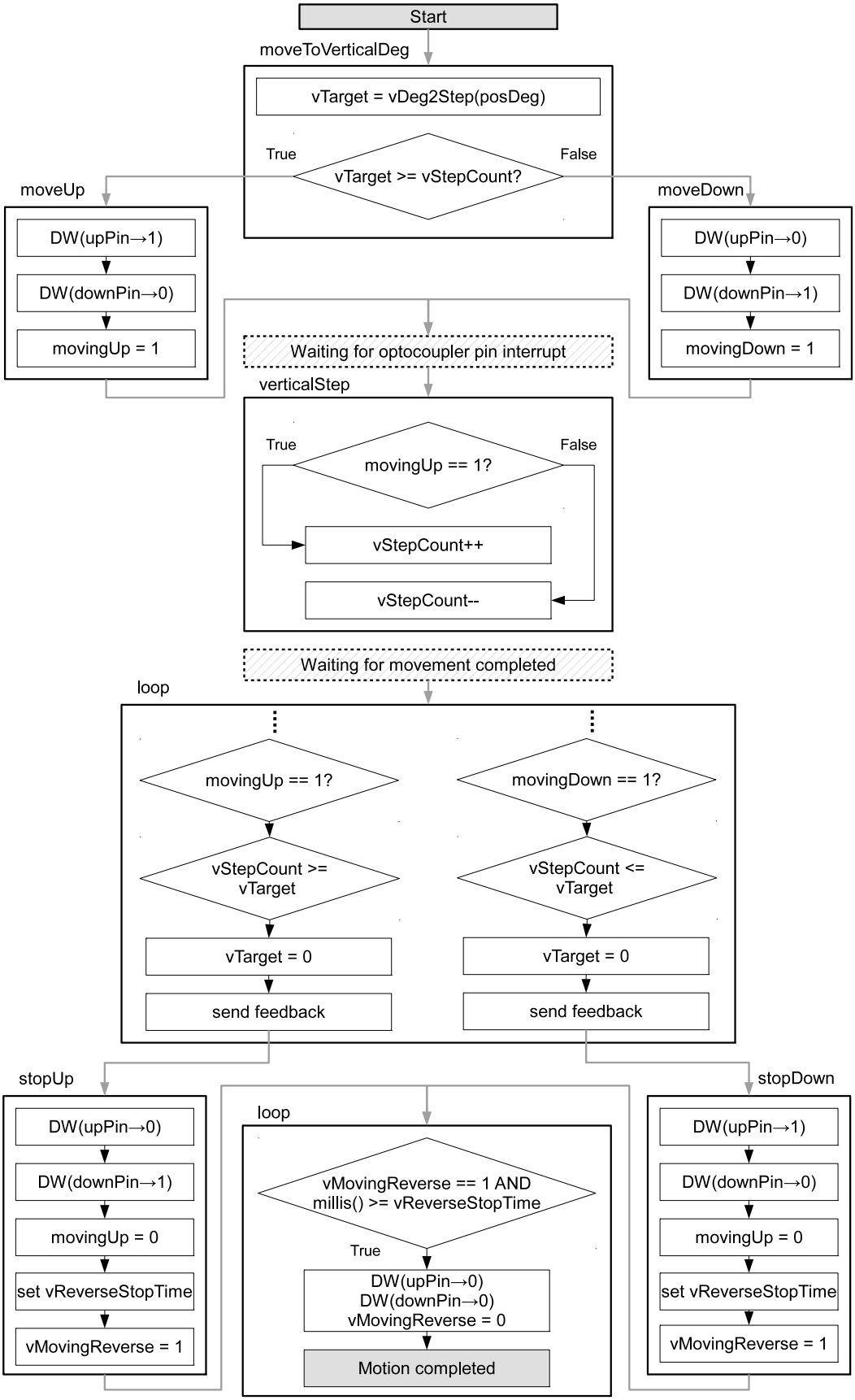


Figure 30: Motion control diagram ("DW" refers to `digitalWrite()` function)

7.3 Control Unit

The main task of the control unit is communication with remote devices (usually smartphone over Bluetooth), sending commands to the head and receiving events from the head. It also controls the display and it serves as a clock source for the busses. Most important feature of the control unit is the process of creating panorama or high resolution images. Because the images are captured in orthogonal rows and columns, the feature is often referred to as Grid feature.

7.3.1 Panorama Process (The Grid Feature)

The process of capturing the grid is quite complex and it is represented by the flowchart on Figure 31. Before the beginning, some information is obtained: vertical and horizontal range of the panorama, overlap sizes, number of exposition for each position (for HDR) and shutter time and focal number for each frame position and exposition. All the properties are received via Bluetooth and stored to global variables. Commands are described in chapter 7.4.

When all the parameters are collected, grid process may begin by calling `startGrid()` function. At first the function detects the focal length of used lens from camera over PTP protocol or if this information is not available it can be manually typed in the user interface of the application. The size of the sensor is also loaded and angle of view of the camera are calculated. Next step is calculating of frame overlaps. Overlap size is received as a number ranges from 0 to 100 and it represents the percentage of the angle of view of the frame. The default value of the overlap is 1/5 of the frame size. When all the information is available the size of the grid (number of rows and columns) will be calculated. There are two variables `gridActualFrameV` and `gridActualFrameH` which keep information about actual horizontal and vertical frame position. At the beginning, both these variables are set to one. There is also `actualFrameNum` variable which keeps the number of frame that is actually being captured. Flag `firstFrame` is set to `TRUE` and `initFrame()` function is called. There is global variable `frameStatus` which keeps state of the frame and takes values from 1 (new frame is initialized) to 11 (grid process completed). The statuses are described in Table 4. Now, actual position in degrees is calculated with the knowledge of the vertical and horizontal frame number. Flags `moveVReady` and `moveHReady` indicates if the motion to desired position is completed. Both flags are set to `FALSE` and functions `vMove(float angle)` and `hMove(float angle)` are called. They accept the appropriate desired position in degrees as an argument and they send command I²C to the head.

Status	Description
1	Next frame initialized
2	Frame position calculated, ready for motion
3	Move command sent, waiting for feedback
4	Motion finished
5	Exposure index set to 1, ready to start capturing in this position
6	Ready to capture image
7	Capture command sent, waiting for event from camera
8	Capture completed
9	Ready to next frame position
10	Preparing next frame
11	Process finished

Table 4: Statuses in grid process

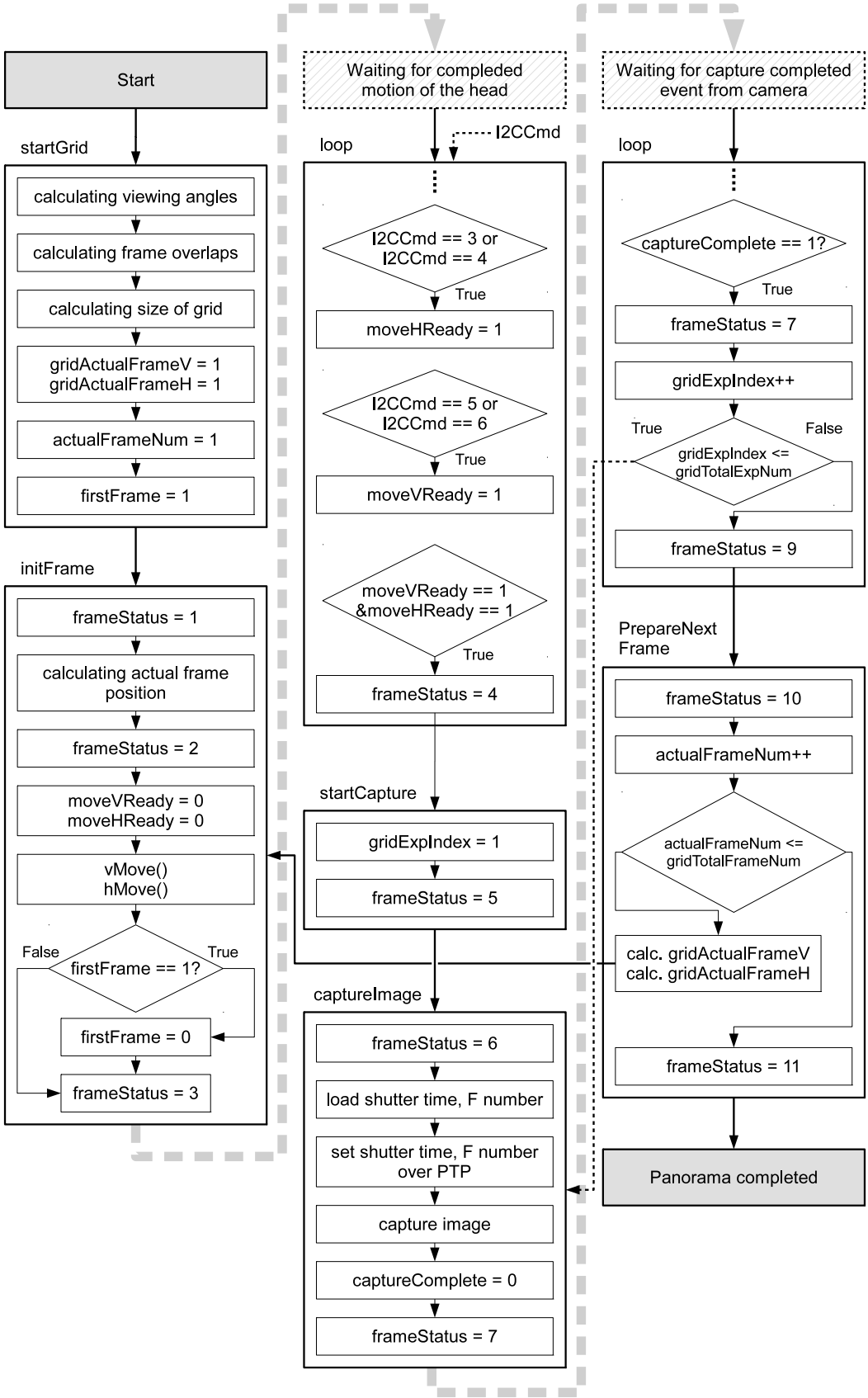


Figure 31: Grid process diagram

Then the system waits until I²C event from the head, which indicates that the movement is finished, is received. If this happens, flag `moveHReady` resp. `moveVReady` is set to `TRUE`. When the movement in both directions is completed and flags are in `TRUE` state, we can deal with capturing. Function `startCapture()` is called and variable `gridExpIndex` is set to 1. This variable keeps the index of actual exposure. This is important in case of HDR imaging because more than one exposure is realized for each frame position. Exposure properties are loaded in `captureImage()` function and camera is configured by PTP command. Capture command is also sent and flag `captureComplete` is set to `FALSE`.

System waits for the event from the camera that the capture was completely finished and simultaneously flag `captureComplete` is set to `TRUE`. This is detected in `loop()` function and if this occurs, `gridExpIndex` variable is incremented. Now, if actual exposition number `gridExpIndex` is less than total exposition number `gridExpTotalNum`, the system will take another image in the same position but with another exposure setting. Otherwise the head will move to next position. Actual frame number is incremented and if the number is less than or equal total frame position, horizontal and vertical index of new frame position is calculated and `initFrame()` function is called. Otherwise, all frames are finished and process is completed.

The order of position of the individual frames is illustrated in Figure 32. The logic is based on series of conditions working with indexes of rows and columns. This order was evaluated as the most time-efficient.

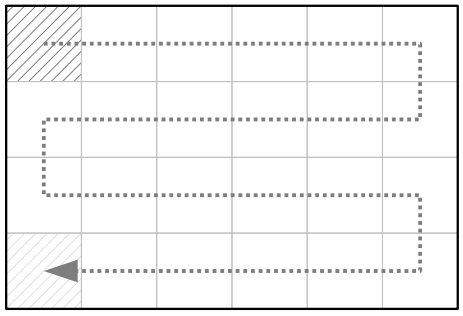


Figure 32: Order of the motion of the head in panorama scene

7.3.2 Camera Control

Communication between camera and control unit is realized via PTP[26]. The protocol allows data transfer between multimedia devices, computers and other devices. In addition, many modern digital cameras can be controlled via PTP from a USB host device. This can be computer, smartphone, Arduino or other devices. The communication takes place over a USB connection. For getting and setting camera features, devices are expected to be in synchronous Bulk Transfer Mode. Camera events are sent back to the host device via the USB asynchronous interrupt endpoint.

The implementation of these features on Arduino platform consists of two parts. The first is driver for USB Host Shield itself and the second for PTP. There are few libraries for this purpose. USB Host Shield library [38] is used for the device. It is developed by Oleg Mazurov and the team who realize the project Circuits At Home. The code is released under the GNU General Public License. Revision 2.0 of MAX3421E-based USB Host Shield Library for Atmel AVR's is used. For camera control, Arduino Camera Control library [39] by the same developer team is utilized. This library depends on Host Shield Library and supports Canon EOS, Powershots and Nikon DSLRs.

[26] PTP – Picture Transfer Protocol

There are four libraries that must be included for proper function. `usbhub.h` provides Host Shield communication, `ptp.h` allows sending commands to camera via generic PTP and libraries `nikon.h` and `nkeventparser.h` allows the capturing of camera events.

USB must be initialized in `setup()` function by the `Usb.Init()` command. In the `loop()` function, there is `Usb.Task()` command which calls USB library and check for new events, camera states, data to transmit etc. To start using the library, new class must be created. The class inherits from `PTPStateHandlers` class which is defined in PTP library. The class may look like this:

```
class CamStateHandlers : public PTPStateHandlers {
   enum CamStates { stInitial, stDisconnected, stConnected };
   CamStates stateConnected;

public:
   CamStateHandlers() : stateConnected(stInitial) {};

   virtual void OnDeviceDisconnectedState(PTP *ptp);
   virtual void OnDeviceInitializedState(PTP *ptp);
} CamStates;
```

Two methods are declared and they represent initialized and disconnected state of the camera. Both these methods `OnDeviceDisconnectedState(PTP *ptp)` and `OnDeviceInitializedState(PTP *ptp)` are defined immediately. `*ptp` represents the reference to PTP class and allows the function to work with the implementation of the protocol. The PTP library is called in each loop of the `loop()` function. Camera is checked and one of the state representing methods is called. Disconnected state is just for information but initialized state is much more interesting because PTP commands may be sent. Sending of commands is realized such that if we want to capture image somewhere in code, flag `captureReady` is set to `TRUE`. Then, when `OnDeviceInitializedState` method is called, the flag is tested and in case of `TRUE` state, the block with PTP commands is executed and flag is set back to `FALSE` state. Sending commands is realized by this code:

```
void CamStateHandlers::OnDeviceInitializedState(PTP *ptp) {
   …
   if(captureReady) {
      captureReady = false;
      ptp->OpenSession();                                    // open ptp session
      ptp->SetDevicePropValue(0x500D, (uint32_t)captureExpTime); // set shutter speed
      ptp->SetDevicePropValue(0x5007, (uint32_t)captureFNumber); // set F number
      ptp->CaptureImage();                                   // start capture
   }
}
```

Three methods are used in the code above. `OpenSession()` starts the PTP session with the camera. `SetDevicePropValue (const uint16_t pcode, uint32_t val)` allows to set property of the camera. The first argument is the code of the property and second argument is a value. When camera is configured, `CaptureImage()` is called which starts capturing. List of codes is not universal for all cameras but one set of codes is usually in usable for all products of the same vendor.

Receiving of events from the camera is little more complicated. We have to prepare the class for event handling. This class is inherited from `NKEventHandlers` and it has virtual method `OnEvent(const NKEvent *evt)` that is called when event is added to the object of the class.

```
class NKEventDump : public NKEventHandlers {
   public:
      virtual void OnEvent(const NKEvent *evt);
};
void NKEventDump::OnEvent(const NKEvent *evt) {
   switch (evt->eventCode) {                  // get code fo the event
```

```
   case 0x400D:                                  // capture completed
      captureCompleted = true;                  // set flag
      break;
   };
}

NKEventDump  dmp;                                // create object
```

Now, we have to create class which inherits from `NikonDSLR`.

```
class Nikon : public NikonDSLR {
   …
}
```

The class has virtual method `Poll()` that provides periodical checking for camera event. If event is received, it is stored to `dmp` variable which if object of the `NKEventDump` class. `OnEvent` method gets the event code and if "capture completed" code is received, flag `captureCompleted` is set to `TRUE`.

There was an issue with timing and it takes a long time to debug. The result is the following finding: If the period of event checking is too short, communication conflict between the Host Shield and camera occurs. This manifested so that camera took three images and then communication stopped working. For final setting, period of checking the events was set to 300 ms. This value was successfully tested and no problems occurred.

7.3.3 Display

The display is used to displaying information about the status of the control unit. Some screenshots of the display are in Figure 33. The resolution is 128×64 points. Top 16 rows of pixels are yellow and the rest of the display is blue but whole the matrix is controlled as it was one continuous display.

Figure 33: Display with three screens. Left image shows co-called pre-grid mode which runs before grid process starts and shows actual focal length and focus mode. Central image shows calibration mode allowing set the head to the correct position. The right image is the screen of the progress on grid mode.

Display control is provided by Adafruit_SSD1306 library [40] and Adafruit_GFX library [41] [42] which provides a common syntax and set of graphics functions for many LCD and OLED displays. Usage of the library is illustrated in following code:

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

static const unsigned char PROGMEM WaveArt[] = {         // define bitmap
   0x30, 0x38, 0x30, … 0x1F, 0x0C, 0x00,
};

void setup() {
   …
   display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C addr 0x3C
   …
}
```

```
void loop() {
   display.clearDisplay();                                  // clear display
   display.drawBitmap(16, 22, WaveArt, 96, 17, WHITE);      // draw bitmap
   display.setCursor(13,0);                                 // set cursor to coordinates
   display.setTextSize(1);                                  // set text size
   display.setTextColor(WHITE);                             // set text color
   display.println("Iitializing");                          // write text
   display.display();                                       // display the graphics
}
```

Display object is created at the beginning and initialized in the `setup()` function with address of the display. After that we can use function to draw bitmaps, write text and set properties. Commands are described in comments in the code above.

7.3.4 Accelerometer

Two libraries are used for communication with the accelerometer. The first is Adafruit_Sensor library [43] which provides sensor driver and Adafruit_ADXL345_U [44] allowing the communication with ADXL 345 accelerometer. Following example shows the usage of the library:

```
// Accelerometer
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>

Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);  // create object
void setup() {
   …
   accel.begin();                               // start using sensor
   accel.setRange(ADXL345_RANGE_4_G);           // set sensitivity of the sensor
   accel.setDataRate(ADXL345_DATARATE_50_HZ);   // set sampling frequency
}

void loop() {
   sensors_event_t event;
   accel.getEvent(&event);                // register event

   float x = event.acceleration.x;        // get value of acceleration ix x axis
   float y = event.acceleration.y;        // get value of acceleration ix y axis
   float z = event.acceleration.z;        // get value of acceleration ix z axis
}
```

The values of acceleration are displayed on the display. Last three values are stored in array which works as shift register with FIFO[27] principle. The variance of these three values is calculated in real time and if variance is larger than threshold, head is moving and warning is displayed.

7.3.5 Sound Signalization

Sound signalization is supported for better indication of some states. The unit beeps when capture of each frame is completed and when panorama process is finished. `tone()` function is used to generate the ton. It accepts three arguments: output pin, tone frequency in Hertz and tone duration in milliseconds. Frequency of 2 kHz was used after subjective tests. For repeating tones, Simple Timer library is used. Its function `setTimer()` allows to execute the callback after specified time and specified number of repetitions.

```
#include <SimpleTimer.h>
SimpleTimer sTimer;
```

[27] FIFO – First In First Out

```
void beep(int toneMs, int pauseMs, byte repeat) {
  toneDuration = toneMs;
  sTimer.setTimer(toneMs+pauseMs, playTone, repeat);
}
void playTone() {
  tone(beepPin, 2000, toneDuration);
}
```

7.3.6 Code Analysis

The code of the control unit has about 900 lines and size of binary sketch is about 50 kB (approx. 19 %). The sketch uses 3826 Bytes of SRAM (approx. 48 %).

7.4 Communication Interfaces

This chapter describes the structure of commands between individual modules.

7.4.1 Control Unit to Head I²C Command

The structure of these commands is specially adapted for sending information to the head about desired position of motion and appropriate direction. We need to send three parameters: information that says which one of vertical or horizontal motion will be realized, the sign of the movement from the zero position (negative values are used for left and down movement and positive values for up and

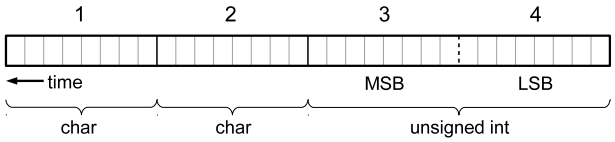


Figure 34: Structure of command of the control unit to the head

right direction) and the angle. The command consists of 4 bytes. First byte specifies V/H movement, the second byte specifies the direction. Third and fourth bytes are MSB and LSB of unsigned integer which defines the angle. Angles are calculated and stored as a type of float. Because sending of floats would be quite complicated, the angle in float is multiplied by 100, rounded and stored as unsigned integer. This 16 bit representation is simple to divide to two bytes and send separately. Inverse opera-

Byte number	Code	Meaning
1	10	Horizontal motion (pan)
	11	Vertical motion (tilt)
2	1	Positive direction
	0	Negative direction
3	-	MSB
4	-	LSB

Table 5: I²C commands for controlling the head

tion is realized in the head to get value in degrees. Structure of the command is illustrated on Figure 34. Codes of individual bytes are in Table 5.

For example command (10,0,1,244) is illustrated on Figure 35 and will be interpreted as "move to position -5 degrees in horizontal direction".

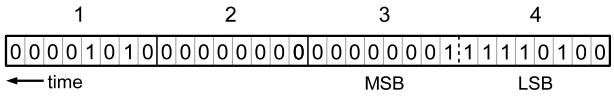


Figure 35: Example of command from the control unit to head

7.5 Head to Control Unit I²C Events

Events from the head are sent to inform the control unit about internal states, completed tasks etc. Only two bytes are needed for this purpose. First byte is the code and holds information about the type of event. Second byte is the parameter which describes specific event. Codes and parameters are described in Table 6.

First Byte Code	First Byte Meaning	Second Byte Code	Meaning
1	Success	1	Vertical calibration completed
		2	Horizontal reset completed
		3	Left motion completed
		4	Right motion completed
		5	Up motion completed
		6	Down motion completed
		7	Head connected
2	Error	1	Motion error
3	Info	1	Top end stop reached
		2	Bottom end stop reached

Table 6: Codes of head event

For example event (1,5) will be interpreted as "motion up successfully completed".

7.5.1 Bluetooth Commands

Bluetooth commands are realized as series of ASCII characters. There are two types of commands. The first are commands for system default configuration and second is the group of command for setting panorama feature. Commands are described in Table 8 and Table 7.

Command	Meaning	Parameters
gridstart	Start loading grid command	-
gridrange h_range,v_range	Set vertical and horizontal range of the panorama	h_range: horizontal range (0°,360°) v_range: horizontal range (0°,60°)
gridexp time,f_number	Add an exposure	time: code of exposure time f_number: code of exposure f number
gridsensor width,height	Set dimensions of sensor	width: sensor width in µm height: sensor height in µm
gridoverlap overlap	Set overlap of neighboring frames	overlap: overlap in % (0-100)
gridfinish	Command loaded, start the process	-

Table 7: Series of Bluetooth commands for panorama process

All parameters are of type unsigned int and they are sent in decimal format. Each command starts with "!" character and ends with "$" character (without the quotes). The list of codes for exposures times and focal numbers is included in appendix.

Command	Meaning	Parameters
beepon	Enable buzzer	-
beepoff	Disable buzzer	-
pregridstart	Start pre-grid mode	-

Table 8: Bluetooth command for device configuration

7.5.2 Bluetooth Events

Bluetooth events have the same structure as I²C commands used for communication between Control unit and head. It consists of four byte. First two bytes are interpreted separately and last two bytes are MSB and LSB of the second parameter. The structure is illustrated on Figure 34. First byte determines the type of event (1 – success, 2 – error, 3 – info), second byte is first parameter (8-bit) and last two bytes are second parameter (16-bit). Table of codes is included in appendix.

7.6 Android Application

The application provides comfortable configuration of capturing process. It is developed with help of MIT[28] App Inventor [45]. This is powerful tool for graphical programming of application for devices with Android operation system. Creating of the application has two main parts represented by two modes: Designer and Block mode. In Designer mode components are dragged from pallets to canvas of the application. Also non visual component can be added like timer, Bluetooth client etc. When the components are placed, their properties can be set in GUI[29] panel.

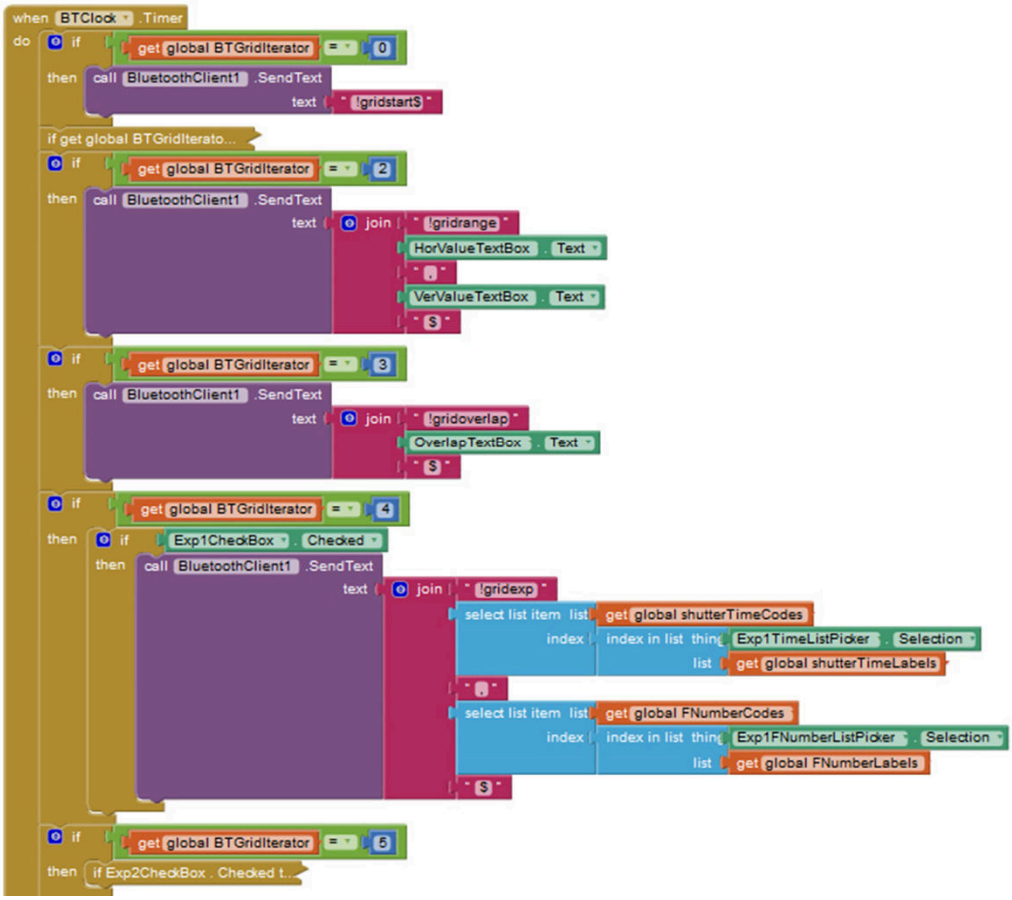


Figure 36: Sample block of code that provides sending commands via Bluetooth to the control unit

After initial configuration of components, Block mode is activated and behavior of the components is defined. The software provides many features needed for common programming – control structures, mathematical and logical operation, procedures, events etc. Sample block of the code is illustrated in Figure 36. Detailed information can be found on the web pages of the project.

Our application is divided to several parts. The first provides Connecting and disconnecting Bluetooth module to the control unit. Both devices must be paired before the use of this application and Bluetooth must be activated else alert with the warning is displayed. Control unit is selected from the list picker and connection is automatically established. List picker turns green and disconnect button is activated in case of success. Third button in top line labeled "CAL" is for leveling of the tripod and head to the correct position. Next sections are for exposure settings.

[28] MIT - Massachusetts Institute of Technology
[29] GUI - Graphical User Interface

First part defines frame overlaps in percentage. Next section defines size of the sensor in μm which is needed for calculation of angle of view. Last pair of textboxes is for setting horizontal and vertical range of panorama in degrees. Last section is for exposure setup. It is possible to activate up to 6 exposures for each position of frame. Parameters of each exposure may be set independently. The first exposure is required and cannot be deactivated. All other ones must be activated by the checkbox and exposure time and focal number must be selected. Then all the parameters are set, process may be started by Start button.

Figure 37: Screenshot of Android control application

All the commands respects the structure which is described in Table 7. For successful transmission of commands via Bluetooth, small delay is applied between each command. Delay of 20 ms was determined as sufficient. Screenshot of the application is illustrated on Figure 37.

7.7 Image Stitching

Image stitching is realized in MATLAB. Big advantage of this solution is ease of installation, high efficiency of software development and wide bundle of available algorithms.

The process starts by loading images. This is realized by `imageSet()` function which accepts directory with the set of images. They are displayed by `montage()` function.

Now we have to detect features and register image pairs. Images are sequentially processed and registered. Firstly, we detect and match features between $I(n)$ and $I(n-1)$. Then we estimate the geometric transformation T(n), that maps $I(n)$ to $I(n-1)$. Finally, we compute the transformation that maps $I(n)$ into the panorama image as $T(n) * ... * T(n-1) * T(n)$. The process begins by converting the image to gray scale. SURF detection algorithm from Computer Vision System Toolbox is applied and features are extracted:

```
grayImage = rgb2gray(I);
points = detectSURFFeatures(grayImage);
[features, points] = extractFeatures(grayImage, points);
```

Then transformation matrix is initialized and projective transform is applied. The matrix keeps information about geometric transformation of each frame. After features of the first frame are extracted, for loop over all frames is proceed. It extracts features of next frame and estimates geometric transform based on extracted points. Estimated matrix is stored to vector of transform matrixes. In the end, the matrix is multiplied by all previous matrixes.

```
% Estimate the transformation between I(n) and I(n-1).
tforms(n) = estimateGeometricTransform(matchedPoints, matchedPointsPrev,...
        'projective', 'Confidence', 99.9, 'MaxNumTrials', 2000);

% Compute T(1) * ... * T(n-1) * T(n)
tforms(n).T = tforms(n-1).T * tforms(n).T;
```

At this point, all the transformations are relative to the first frame. It causes that frames on the end of the row are strongly destroyed. A nicer panorama can be created by modifying the transformations such that the center of the scene is the least distorted. This is accomplished by inverting the transform for the center image and applying that transform to all the others. We start with the `projective2d outputLimits()` function that finds output limits for transform and we use these limits to automatically find the image in the center of the scene. Finally, we take transform of center image and invert it. This inverse transform is then applied to all other transforms.

Next step is initialization the panorama. The panorama is empty at the beginning and all images are mapped into it. We calculate its dimensions with help of `outputLimits` function.

Finally, we use `imwrap` function to map images into the panorama and `vision.AlphaBlender` to overlay images together [46].

Stitching is implemented just for images that are arranged either in a row or column. This is caused by high complexity of stitching images which are arranged in a matrix. For these purposes, open source and commercial stitching software is used and compared in conclusion section. This implementation shows basic stitching process and its results are also presented in conclusion section.

Chapter III Results and Summary

Main goal of the thesis was to realize automated system for capturing panoramic high resolution images. The project consists of several parts: design of mechanical and electronic components, programming of microprocessors in the devices, development of android application and implementation of stitching algorithms. These building blocks will be evaluated separately.

8 Review of Individual Parts of the Work

8.1 Panoramic Head

The panoramic head is mechanically based on commercial solution but electronics was completely redesigned. It provides motion range ±15° in vertical and 360° in horizontal direction. The head can rotate over again. The only potential problem are cables between the head and control unit. The deviation in the horizontal motion is about ±3° on 360°. This can be expressed as ±0.0083° on 1° which is absolutely sufficient. However, mechanical construction causes small clearance about 2°. It does not lead to the complications, but it may be reflected as a small inaccuracy when head changes its direction from left to right or vice versa. Vertical movement does not suffer from such a similar clearance. The dependency of vertical angle on desired number of steps on the code circle is illustrated on Figure 38.

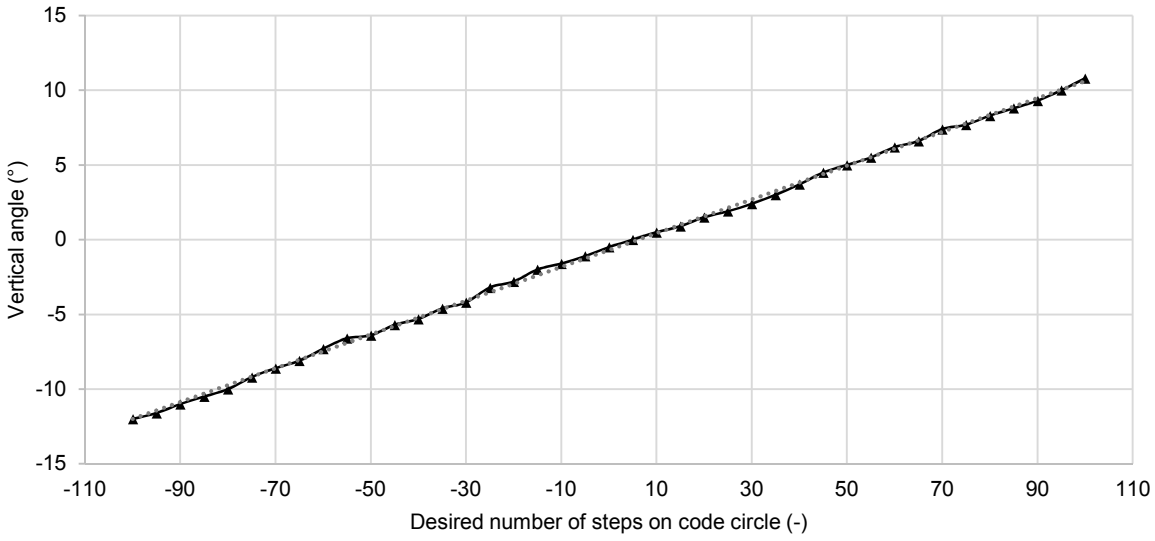


Figure 38: Dependency of vertical angle on desired step on code circle

The Curve in Figure 38 was obtained by measuring with calibrated accelerometer built in smartphone and application that display tilt angle. We can see that the dependence is very close to linear. Deviations from linear curve are caused by detection of pulses coming from the phototransistor of the optocoupler. The code circle is not absolutely ideal as well as relative position of the optocoupler and stripes on code circle. Because of lack of space in the head, the diameter of the circle must be relatively small as well as width of each stripe. The result is that the angle of view of the phototransistor is larger than width of one stripe. It leads to occasional bad interpretation of input signal by interrupt software. Similar situation occurs in horizontal direction but the deviation of the curve was closer to

linear. Horizontal positioning was measured by laser fixed on the head. The laser lit to the marks with angles around the head. Despite the issue described above, the device works well. There were no problems with the motor control and also immediate stop of motors works well. Device is very silent and fast enough.

Although the results are good, the position detection and motor control of the head are unnecessarily complex. A better solution would be to use stepper motors where no position detection is needed. Motors could be controlled by the same PUSH-PULL motor driver which is currently used. This fact was already known when selecting the appropriate mechanical solution of the head, but there was no other available and affordable solution for this reason. It also was not possible to replace motors in the currently used head because of mechanical dimensions and construction of gears. Stepper motors would provide very high accuracy and they would also simplify the whole position control process. The best solution would be to build very robust head with minimal clearance, 360° vertical range and at least 60° horizontal range. The motion would be realized by the pair of stepper motor controlled by PUSH-PULL driver and microcontroller as it is now.

8.2 Control Unit

Control unit works exactly as planned. Communication with all interfaces works correctly and no problems with loosing connection or timing inaccuracy were found. Good results were given by the display. It has excellent readability on direct sunlight. Very useful is screen with information about the progress of panorama capturing which is launched when process begins. It displays actual frame and number of total frames, number of rows and columns, focal length, motion range and actual row and column. The use of the dural/plastic case has proved to be very good choice because the device does not heat on direct sunlight (compared to all-plastic box of battery pack). Bluetooth connection is not blocked by the box and works well. Communication with the camera including sending commands and receiving events also works correctly. Initially it was considered to add some additional control buttons to the control unit, but it turned out that it is not needed and remote control via smartphone is comfortable enough. For debugging purposes, USB connector is preserved and control software can be easily updated. Next very useful feature is piezo buzzer which indicates all important events of the head. It beeps when system is ready after initial calibration, when each frame is captured and when the whole set of the images in the panorama is completed. Each task has different length and different number of repetition of the beep.

8.3 Control Application

The control application was also proved to work well. Its screen is shown in Figure 37. Firstly, Bluetooth must be enabled by the smartphone or tablet and the head must be paired with the device. Then the application allows to select the head from the list and connection is established. Now, application can be used and we can set the parameters. It allows to set dimensions of image sensor, frame overlaps and motion range of panorama in angles. These properties are used to compute positions of individual frames, number of rows, columns etc. The most important section of the application is set of the exposure setting. It currently allows to activate up to six exposures, only the first one is activated automatically. Each exposure is set by exp. time and focal number. This feature can be used for HDR processing. The application also provides to enable calibration mode, which shows tilt leveling of the head and the level of head shaking on the display. This is useful feature when tripod is leveled. At the end, the application allows to interrupt current operation and set the head to the initial position. The application is universal for any device with Android operation system and it does not require any special features of the device. The big advantage of the application is the fact that the Bluetooth connection

may not be preserved throughout the duration of capturing process. The critical part is at the beginning when initial commands are sent to the control unit and then all properties, positions and exposure settings are computed and processed by the control unit. This is very useful when we have long lasting set of exposures and Bluetooth connection is lost. Due this logic, the whole process is completed with no interruption. For future improvements, the application should be developed by standard way in java because it offers more features and adjustments than currently used MIT App Inventor. New version of the application should also allow to add more exposures and simply setting of exposure bracketing.

8.4 Panorama Stitching

Last part deals with panorama stitching. The main goal was to implement solution for image stitching and verify the effectiveness of used algorithms. The software was implemented in Matlab environment which is ideal for prototyping and image processing. The script starts with GUI to choose a directory with the set of images to stitch. Source images are displayed in a grid and then all images are sequentially processed. When the process is completed, panorama is displayed and saved. The process is very time-consuming and it was necessary to scale down source images to about 3-4 Mpixels because the time of processing of the original images (12 Mpixels) was too long or the script went out of memory. The process consists of three main steps: aligning images, geometrical correction and image blending. Aligning of images works well, but sometimes, geometrical correction of some fames is computed slightly inaccurately. This occurs on images with repeating pattern or smoothed object. Typical example is the sky with clouds which has no sharp edges. In this case, feature points are not detected correctly and they are placed to different position on the same scene on neighboring images. This is reflected in the transform matrix and then in geometrical transform of some images. This causes that visible edges may appear between images in places with problematic object in the panorama. Finally, image blending process works fine. This phenomenon also appeared while stitching of "gigapixel" photography in commercial software Photomatix. It was not able to detect feature points in the whole sky with clouds and all these frames (about 35) must have been marked manually. Possible solution would be to use alternative feature detector or modification of SURF detector. Because of wide focus of this work, there was no time to noticeably improve stitching algorithm and it could be the point of view for further research. Comparison of panoramas stitched from the same set of source images is in Appendix A-I.

The problem of processing of high resolution panorama images is extremely high computational complexity and all algorithms must be optimized for high performance. Similar complication is huge amount of data. Sample hi-resolution image was composed from 126 source images whose size is about 500 MB. It was verified, that it is best to capture images in manual mode with fixed exposure settings as well as wit fixed manual focus. Changing these properties while capturing may cause interesting effects but it does not usually lead to good results. We can minimize geometric distortion by using lens with a long focal length. The longer focal length, the smaller distortion. Practically, focal length which is longer than 100 mm (with sensor dimensions 23.6 x 15.8 mm) provides good conditions and relatively small distortion.

Sometimes high resolution photography is said to be pointless because the images are presented (printed or displayed) on relatively small areas. This is partly true, but down-scaled high resolution images have much better sharpness without additional processing and also noise level is noticeably lower compared to common images.

9 Sample Photos

There are three sheets with photos that were captured using my panorama system. They are included in appendix A-I, A-II and A-III.

The first sheet (Appendix A-I) demonstrates the results of stitching process with different software. The panoramas were compounded from 9 images in a row. The first image is result of author's script written in Matlab, the second was processed with open-source Hugin Panorama Photo Stitcher. Last two images were stitched with commercial software ArcSoft Panorama Maker and Adobe Photoshop. We can see, that last two results are very similar both in colors and precise of stitching. Hugin software gives also good results but it applies slightly different projection model which is noticeable on the forest on left side of the image. This software also have a bit different color reproduction – it makes panoramas cooler. The first image contains some visible edges in upper region of image. This is caused by the problem described above. Color reproduction is similar to commercial solutions.

The second sheet (Appendix A-II) shows high resolution image. It was compounded from 126 source images arranged in 9 rows and 14 columns. The size of each source image is 4288x2848 pixels. Total size of the source images is about 500 MB. Panorama was stitched with PTGui software. There was a problem with aligning of images containing the sky because the algorithm was not able to detect feature points and all the points of the sky must have been marked manually. Final panorama looks very good, sharp and contains minimum artifacts. Only some trees at the bottom of the images are little blurred. This is caused by the fact, that camera was focused somewhere to horizon and trees near the camera were moving because of gusty wind. Below main panorama, there are two cutouts in 100 % magnification (projection 1:1). Also these cutouts looks good. The source images were captured by DSLR Nikon D90 with the lens Sigma 70-300 mm 1:4-5.6 APO DG. Focal length was set to 300 mm which is equivalent to 450 mm on full frame sensor. Exposure time was 1/250 s and focal number 9.

The last sheets shows the results of HDR imaging. The camera was controlled by the control unit. Both images are compounded from 19 source images of the same scene with different exposure setting. Exposure time changed from -3 EV to +3 EV with the step of 1/3 EV. Because the Android application allows to set up to 6 exposures, capturing of these HDR images was controlled by laptop over USB cable. Images were processed by Detail Enhancer and Tone Compressor algorithm in Photomatix software.

10 Conclusion

The aim of the work was achieved. System for acquisition of panoramic photos with high resolution was designed and realized. It includes mechanical and electronic design of the panoramic head, control unit and battery pack. These units were programmed to communicate with each other and with other devices. System control is realized by Android application which communicates with the control unit via Bluetooth. Image stitching algorithm was implemented in Matlab environment. Although this software works slightly inaccurately, it can be used as a demonstration of stitching process. Production stitching software requires long time development and high performance optimization. High resolution panorama images were obtained and results are presented in Appendix. The improvement of my solution could be redesigning of the panoramic head to solution using stepper motors for higher accuracy and simplicity of position control. This work could be also followed by deeper research of image stitching issue.

Panoramic and high resolution imaging has many pitfalls and difficulties, but it provides amazing results and experience.

Chapter IV Lists

11 References

[1] LI, Jubiao a Junping DU. Study on panoramic image stitching algorithm. *2010 Second Pacific-Asia Conference on Circuits, Communications and System: 1 - 2 Aug. 2010, Beijing, China*. Piscataway, NJ: IEEE, 2010, (10). DOI: 10.1109/paccs.2010.5626602. ISBN 9781424479689.

[2] GIGAPAN SYSTEMS. *GigaPan EPIC* [online]. 2013 [cit. 2015-04-05]. Dostupné také z: http://gigapan.com/cms/shop/store

[3] SYRP. *Syrp Genie* [online]. 2013 [cit. 2015-04-05]. Dostupné také z: https://syrp.co.nz/products/genie

[4] GONZALEZ, Rafael C a Richard E WOODS. *Digital image processing*. 3rd ed. Upper Saddle River: Pearson, c2008, xxii, 954 s. ISBN 01-316-8728-X.

[5] JUAN, Luo a Gwun OUBONG. SURF applied in panorama image stitching. *2010 2nd International Conference on Image Processing Theory, Tools and Applications (IPTA 2010): Paris, France, 7 - 10 July 2010*. Piscataway, NJ: IEEE, 2010, (10): 205-234. DOI: 10.1007/978-1-4302-2842-4_9. ISBN 9781424472499.

[6] HARRIS, C. a M. STEPHENS A Combined Corner and Edge Detector. *Alvey Vision Conference*. 1988. DOI: 10.5244/c.2.23.

[7] BROWN, M., R. SZELISKI a S. WINDER Multi-Image Matching Using Multi-Scale Oriented Patches. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005. DOI: 10.1109/cvpr.2005.235.

[8] ZHANG, Chen, Mengyang ZHAO a Liang YUAN. An improved algorithm for corner detection. *International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011: Harbin, Heilongjiang, China, 12 - 14 August 2011*. Piscataway, NJ: IEEE, 2011, (11): 750-753. DOI: 10.1109/emeit.2011.6024069. ISBN 9781612840888.

[9] MORDVINTSEV, Alexander. Introduction to SURF. *OpenCV-Python Tutorials* [online]. 2014 [cit. 2015-04-10]. Dostupné také z: http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html#surf

[10] WOODRUFF, Caleb. Feature Detection and Matching. *Computer Science & Engineering* [online]. 2013 [cit. 2015-05-4]. Dostupné také z:

https://courses.cs.washington.edu/courses/cse576/13sp/projects/project1/artifacts/woodrc/index.htm

[11] SUN, Shiliang a Rongqing HUANG. An Adaptive k-Nearest Neighbor Algorithm. *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery: FSKD 2010 ; 10 - 12 Aug. 2010, Yantai, Shandong, China*. Piscataway, NJ: IEEE, 2010, (10). ISBN 9781424459346.

[12] MOORE, Anderw W. *An Introduction tutorial on kd-trees*. Carnegie Mellon University, 2004.

[13] CHUM, O. a J. MATAS Optimal Randomized RANSAC. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2008, **30**(8): 1472-1482. DOI: 10.1109/tpami.2007.70787. ISSN 0162-8828.

[14] LI, Yanfang, Yaming WANG, Wenqing HUANG a Zuoli ZHANG. Automatic image stitching using SIFT. *International Conference on Audio, Language and Image Processing, 2008: ICALIP 2008 ; 7 - 9 July 2008, Shanghai, China ; proceedings*. Piscataway, NJ: IEEE, 2008, (8). DOI: 10.1109/icalip.2008.4589984. ISBN 9781424417247.

[15] BESCOR VIDEO ACCESSORIES LTD. *Bescor MP-101* [online]. b.r. [cit. 2015-03-02].

[16] ATMEL CORPORATION. *ATmega48A/PA/88A/PA/168A/PA/328/P - Complete Datasheet* [online]. 2014 [cit. 2015-04-12].

[17] ATMEL CORPORATION. *ATmega48A/PA/88A/PA/168A/PA/328/P - Datasheet Summary* [online]. 2014 [cit. 2015-04-12].

[18] Arduino Pro Mini Specification. *Arduino* [online]. b.r. [cit. 2015-02-14]. Dostupné také z: http://www.arduino.cc/en/Main/ArduinoBoardProMini

[19] STMICROELECTRONICS. *L293b Push-Pull Four Channel Driver* [online]. 2003 [cit. 12.1.2015]. Dostupné také z: http://www.gme.cz/img/cache/doc/399/010/l293b-datasheet-1.pdf

[20] I2C Bus. *Sparkfun* [online]. b.r. [cit. 2015-03-20]. Dostupné také z: https://learn.sparkfun.com/tutorials/i2c

[21] ATMEL CORPORATION. *Atmel ATmega640/V-1280/V -1281/V-2560/V-2561/V - Complete Datasheet* [online]. 2014 [cit. 2015-04-12]. Dostupné také z: http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

[22] ATMEL CORPORATION. *Atmel ATmega640/V-1280/V -1281/V-2560/V-2561/V - Datasheet Summary* [online]. 2014 [cit. 2015-04-12]. Dostupné také z: http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_Summary.pdf

[23] Arduino MEGA 2560. *Arduino.cc* [online]. b.r. [cit. 2015-02-14]. Dostupné také z: http://www.arduino.cc/en/Main/arduinoBoardMega2560

[24] USB Host Shield Specification. *Arduino* [online]. b.r. [cit. 2015-02-14]. Dostupné také z: http://www.arduino.cc/en/Main/ArduinoUSBHostShield

[25] GUANGZHOU HC INFORMATION TECHNOLOGY CO., LTD. *HC-06 Bluetooth Module - Datasheet* [online]. b.r. [cit. 2015-04-14].

[26] Wii Nunchuck as general purpose controller via Arduino board. *Instructables* [online]. b.r. [cit. 2015-01-10]. Dostupné také z: http://www.instructables.com/id/Wii-Nunchuck-as-general-purpose-controller-via-Ard/

[27] *Processing Language* [online]. b.r. [cit. 2015-01-10]. Dostupné také z: https://processing.org/

[28] HAMMOND MANUFACTURING. *1455K1202 Enclosure Datasheet* [online]. 2007 [cit. 1015-02-14]. Dostupné také z: http://www.gme.cz/img/cache/doc/622/972/hlinikova-krabicka-u-ha1455k1202-datasheet-1.pdf

[29] USB.ORG. *USB 2.0 Specification* [online]. b.r. [cit. 2015-01-10]. Dostupné také z: http://www.usb.org/developers/docs/usb20_docs

[30] Serial Peritheral Interface (SPI). *Sparkfun* [online]. b.r. [cit. 2015-03-20]. Dostupné také z: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

[31] STMICROELECTRONICS. *LM317 Voltage Regulator* [online]. 1999 [cit. 12.1.2015]. Dostupné také z: http://www.gme.cz/img/cache/doc/331/022/lm217t-datasheet-1.pdf

[32] NATIONAL SEMICONDUCTORS. *LM2576 Series SIMPLE SWITCHER* [online]. 1999 [cit. 30.2.2015].

[33] SCHERZ, Paul a Simon MONK. *Practical electronics for inventors*. Third edition. b.r., xxv, 1014 pages. ISBN 978-007-1771-337.

[34] Arduino Oscilloscope: Poor man's Oscilloscope. *Instructables* [online]. 2013 [cit. 2015-01-05]. Dostupné také z: http://www.instructables.com/id/Arduino-Oscilloscope-poor-mans-Oscilloscope/

[35] Arduino Software. *Arduino* [online]. b.r. [cit. 2015-03-20]. Dostupné také z: http://www.arduino.cc/en/Main/Software

[36] Wire Library. *Arduino* [online]. b.r. [cit. 2015-03-20]. Dostupné také z: http://www.arduino.cc/en/Reference/Wire

[37] PinChangeInt Library. *Arduino* [online]. b.r. [cit. 2015-04-15]. Dostupné také z: http://playground.arduino.cc/Main/PinChangeInt

[38] MAZUROV, Oleg, Alexei GLUSHCHENKO a Kristian LAUSZUS. *USB Host Library Rev.2.0* [online]. b.r. [cit. 2015-03-20]. Dostupné také z: https://github.com/felis/USB_Host_Shield_2.0

[39] MAZUROV, Oleg. Camera Control Library. *Circuits at Home* [online]. b.r. [cit. 2015-03-22]. Dostupné také z: https://github.com/felis/PTP_2.0

[40] SSD1306 OLED driver library. *Adafruit* [online]. b.r. [cit. 2015-04-15]. Dostupné také z: https://github.com/adafruit/Adafruit_SSD1306

[41] Adafruit GFX Library. *Adafruit* [online]. b.r. [cit. 2015-04-15]. Dostupné také z: https://github.com/adafruit/Adafruit-GFX-Library

[42] BURGESS, Phillip. ADAFRUIT. *Adafruit GFX Graphics Library* [online]. 2014 [cit. 2015-04-19]. Dostupné také z: https://learn.adafruit.com/downloads/pdf/adafruit-gfx-graphics-library.pdf

[43] Adafruit Unified Sensor Driver. *Adafruit* [online]. b.r. [cit. 2015-04-19]. Dostupné také z: https://github.com/adafruit/Adafruit_Sensor

[44] TOWNSEND, Kevin. Adafruit ADXL345 Accelerometer Driver. *Adafruit* [online]. b.r. [cit. 2015-04-19]. Dostupné také z: https://github.com/adafruit/Adafruit_ADXL345

[45] MASSACHUSETTS INSTITUTE OF TECHNOLOGY. *MIT App Inventor* [online]. b.r. [cit. 2015-04-19]. Dostupné také z: http://appinventor.mit.edu/

[46] Feature Based Panoramic Image Stitching. *MathWorks* [online]. b.r. [cit. 2015-04-26]. Dostupné také z: http://www.mathworks.com/help/vision/examples/feature-based-panoramic-image-stitching.html

12 List of Figures

13 List of Tables

14 List of Attachments

Appendices

APPENDIX A–1

COMPARISON OF RESULTS

Panoramas compounded from nine images and stitched by various software
No additional processing was appied (such as color correction or sharpening)

Matlab with author's image processing

Hugin Panorama Photo Stitcher

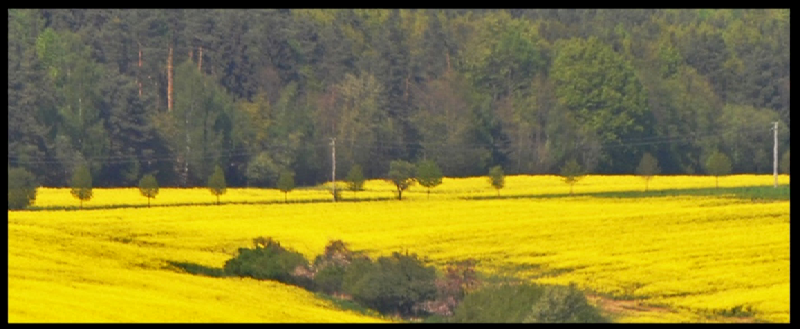
ArcSoft Panorama Maker

Adobe Photoshop – photomerge function

Nikon D90, Hudlice, Central Bohemia 2015

Panorama compounded from 126 images in 14 comulns and 9 rows. Dimensions are 27 956x10 075 pixels (total 281 MPixels). Panorama stitched with PTGui software.

100 % Magnification (1:1)



100 % Magnification (1:1)

HIGH RESOLUTION

APPENDIX A–III

HDR IMAGING

PHOTOS ARE COMPOUNDED FROM 19 IMAGES AND PROCESSED WITH PHOTOMATIX SOFTWARE
CAMERA SETTING WAS MODIFIED REMOTELY BY CONTROL UNIT. EXPOSURES ARE BRACKECTED WITH 1/3 EV DIFFERENCE.

NIKON D80, HUDLICE, CENTRAL BOHEMIA 2015

DETAIL ENHANCER ALGORITHM (PHOTOMATIX) WITH CORRECTIONS IN ADOBE PHOTOSHOP

NIKON D80, VRANNI SKALA, CENTRAL BOHEMIA 2015

TONE COMPRESSOR ALGORITHM AND MAPPING TO GRAY SCALE (PHOTOMATIX)

Appendix B-I

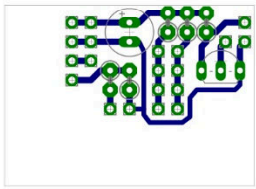
Content of DVD

- **Datasheets**
- **Photo Documentation**
    + Final Devices
    + Development
- **Eagle Schematics**
    + Control Unit Accesory Board
    + Control Unit Connector Board
    + Head Control Board
    + Panorama System
- **Libraries**
    + Arduino
    + Eagle
- **Software & Scripts**
    - Android Application
    - Arduino Skeches
        + Control_Unit_Engine
        + Head_Control_Engine
    - Encoder Circle
    - Matlab
        + Source Images
        Stitcher.m
- **Photos**
    - High Resolution
        + Final Panorama
        + Source Images
    - HDR 1
        + HDR
        + Source Images
    - HDR 2
        + HDR
        + Source Images
    - Panorama+Software Comparison
        + Panorama
        + Source Images

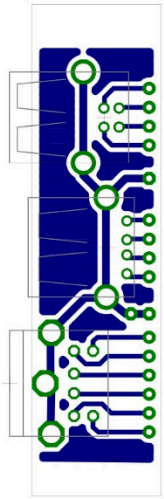
Appendix C-I

Printed Circuit Boards used schemas

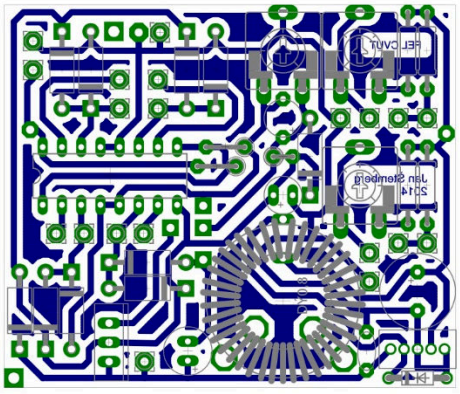
Boards are printed in 100 % size



Board of power module in control unit



Board of module with connectors in control unit



Board of panoramic head