

CHAPTER 3 – RESEARCH METHODS

In this thesis, there are two completely separated subsystems which are RTK GNSS and rotary incremental encoder. Overview of the whole system will be discussed a bit in this chapter. However, explanations for the two subsystems will be covered in detail. The steps taken to construct both subsystems will be presented, including the software and electrical design development as well as experimental results.

3.1 System Overview

In the big picture, the goal is to construct an autonomous car which can determine its current position based on data from satellites and reach the destination accurately by using route data from Google Maps for navigation. The autonomous car will be equipped with RTK GNSS module that always broadcasts its position to the internet by using NTRIP. The advantage of using NTRIP is that user can put base station receiver and rover station receiver as distant as 10km and both receiver data are accessible online. Google Maps is used for navigation to provide the coordinate of destination and the route from the initial position to the destination position. The autonomous car is designed so that it is able to turn left or right automatically by controlling the motor in electric power steering. However, the electric power steering that is currently available in the market is not equipped with a position sensor, therefore it acts like an open loop function. An additional sensor becomes necessary to be installed as it gives feedback to the system so that it becomes closed loop function and inaccuracies can be minimized. Finally, when the autonomous car has reached the destination position, it will stop.

An autonomous car surely consists of many subsystems because there are many factors that should be considered in mechanical, electrical and software design. However, due to time limitation, in this thesis, there are only two specific subsystems that will be covered, which are RTK GNSS and rotary incremental encoder. To further understand the scopes of this thesis, a flowchart is made as can be seen down below. The topics covered in this thesis is typed in red ink.

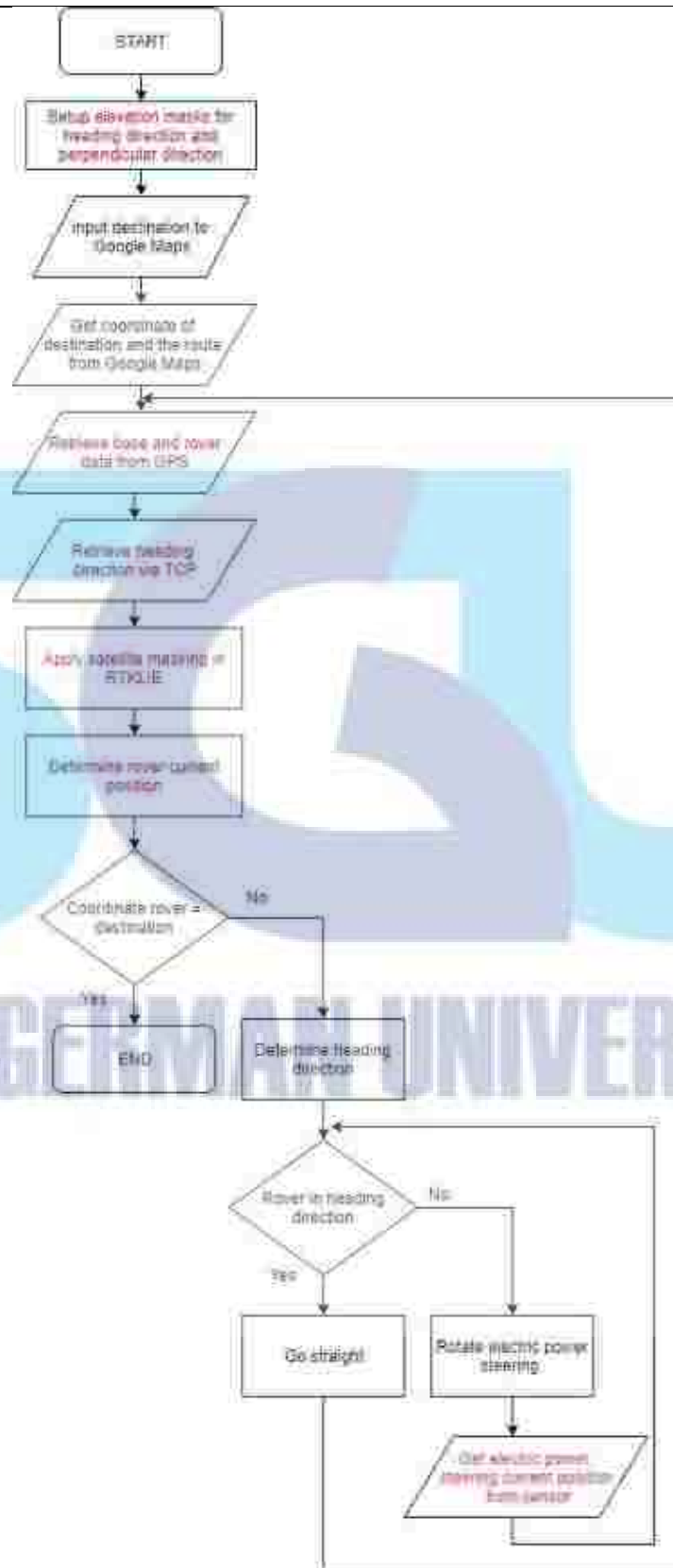


Figure 3.1 System Overview

3.2 RTK GNSS

3.2.1 Base Station Receiver

3.2.1.1 Placement

The base station receiver is placed in the highest point of Imam Bonjol Street, Sukajadi, Karawaci, Tangerang, as there are no obstacles present that can cause multipath. The base station receiver is powered through Micro-USB port.

3.2.1.2 ReachView Settings

When the base station receiver is first connected to the power source, by default, it will create its own hotspot. This is indicated by LED in Reach module that will be flashing white-green color alternately. To set up the base station receiver, connect any device to the hotspot made by base station receiver.

After the device is connected, type 192.168.42.1 or <http://reach.local> on the browser. It will direct the browser to ReachView application. When the ReachView application is fully loaded, the status page is displayed. In the status page, a menu for setting up the ReachView is presented as well as some important information about the Reach performance. This information includes SNR graphs, RTK parameters, positioning mode, solution status, position, velocity, and map.



Figure 3.2 ReachView Base Receiver Status Page

To achieve better performance result, the parameters in RTK Settings must be adjusted. The positioning mode is set to static because the base acts as a reference station for the rover, therefore the base must stand still in its own position. The AR (Ambiguity

Resolution) is enabled for GPS and GLONASS. AR mode is set to fix-and-hold for both GPS and GLONASS because it provides better overall performance than continuous. Elevation mask angle and SNR mask are set to be 0° and the acceleration is set to be 1m/s^2 for both vertical and horizontal axes. The desired satellite constellations to be used is ticked as well as the update rate. In this case, the satellite selection to be used is GPS, Galileo, SBAS, QZSS, and Beidou. However, this selection may vary depends on the geographic location of the receiver and user preferences.



Figure 3.3 ReachView Base Receiver RTK Settings

In Base Mode settings, the corrections output is sent to NTRIP Caster. There are some parameters that should be known prior to sending the correction to the NTRIP Caster. They are IP address, port, password, and mountpoint. In this case, an opensource NTRIP Caster named rtk2go.com with port 2101 is used. The mountpoint is named Reach_Base, however, there is no rule in naming the mountpoint. The password is BETATEST, which is predetermined by rtk2go.

The correction output is sent in RTCM3 format. In this case, the type of RTCM3 messages that are being sent is 1002 (GPS L1 observations), 1006 (ARP station coordinate), 1019 (GPS Ephemeris), 1107 (SBAS MSM), 1117 (QZSS MSM), and 1127 (Beidou MSM). This selection may vary depending on satellite selection that has been previously made in RTK Settings and user preferences. The update rate for all RTCM3 messages is 1Hz.

For the base station receiver position, the coordinates will be taken from the result of averaging the single position for 30 minutes. After the result is obtained, the coordinates can be saved in manual option for future use. Therefore, in a case when the power for base station receiver is cut off but it is still located in the same position, instead of using “average single”, “manual” option can be used.



Figure 3.4 ReachView Base Receiver Base Mode Settings

ReachView supports RTK GNSS data logging. There are four data types that can be logged, which are raw data, position data, base correction, and additional correction. For base station receiver, the necessary data to be logged are raw data. The format selected is UBX, because it takes the least memory than the other format.



Figure 3.5 ReachView Base Receiver Data Logging

In ReachView, there is a GNSS predictor menu, which can help the user to observe the satellites visibility pattern throughout the day. The date and time of prediction can be

set as well as elevation mask. This is useful to help in determining which satellite constellation that can be used at a certain date, time, and location. For example, on Monday, April 23rd, 2018 at 12 PM, in Tangerang area, Beidou, SBAS, and GPS occur more often than the other satellite constellations, therefore those 3 constellations should be considered when determining which satellite constellations to be used.



Figure 3.6 ReachView Base Receiver GNSS Predictor

After the settings are finished, the base station receiver is connected to Wi-Fi so it can broadcast the RTCM3 messages to the NTRIP Caster. In Wi-Fi/Bluetooth settings, connect to the Wi-Fi available. After it is successfully connected, the base station receiver won't create its own hotspot any longer and its IP address will change. In this case, the IP address is changed to 192.168.1.13. Therefore, in the future, to access ReachView, the user must connect to the same Wi-Fi which base station receiver is connected and type 192.168.1.13 instead of 192.168.42.1 on the browser.



Figure 3.7 ReachView Base Receiver Wi-Fi Settings

After the base station receiver is connected to NTRIP, in the status page now there is a street map visible.



Figure 3.8 ReachView Base Receiver Status Page Map

3.2.2 Rover Station Receiver

3.2.2.1 Placement

The rover station receiver placement depends on the testing condition. In this thesis, during a static test, the rover is placed next to the base station receiver, which is in the highest point of Imam Bonjol Street, Sukajadi, Karawaci, Tangerang. During the road test, the rover is placed on top of a car and during combination test with the autonomous car, the rover is placed on the AGV.



Figure 3.9 Rover Station Receiver Road Test Placement



Figure 3.10 Rover Station Receiver Combination Test Placement

3.2.2.2 ReachView Settings

Overall, the settings for rover station receiver is the same as the base station receiver. The only difference is only in the RTK Settings and Base Mode Settings. In RTK Settings, the positioning mode is set to kinematic, because now the rover station receiver wants to get the correction from base station receiver. The AR (Ambiguity Resolution) is enabled for GPS and GLONASS. AR mode is set to fix-and-hold for both GPS and GLONASS because it provides better overall performance than continuous. Elevation mask angle and SNR mask are set to be 0° and the acceleration is set to be 10m/s^2 for both vertical and horizontal axes. The desired satellite constellations to be used is ticked as well as the update rate.



Figure 3.11 ReachView Rover Receiver RTK Settings

In correction input settings, the corrections input is received via NTRIP Caster. The IP address is rtk2go.com with port 2101. The username is rtklib and the password is BETATEST. The mountpoint used is Reach_Base with data type received of RTCM3.



Figure 3.12 ReachView Rover Receiver Correction Input Settings

In Base mode, the settings are pretty much the same as the base station receiver. The only difference is the mountpoint name for the correction output, which is changed RoverTest. The rover station receiver position is not significant in this case, therefore can be neglected.



Figure 3.13 ReachView Rover Receiver Base Mode Settings

3.2.3 RTKLIB

3.2.3.1 Analyzing RTKLIB Features

Due to the various application programs that RTKLIB provided, analysis of each application program becomes necessary to determine the starting point of program modification. By analyzing each application program, the program that needs modification can be decided. All of the application programs can be found in the bin folder of RTKLIB. Firstly, RTKLAUNCH is opened to see all of RTKLIB features. From left to right, sequentially, the RTKPLOT, RTKCONV, STRSVR, RTKCONV, NTRIP BROWSER, RTKNAVI, RTKGET, RTKVIDEO application program launchers are shown.

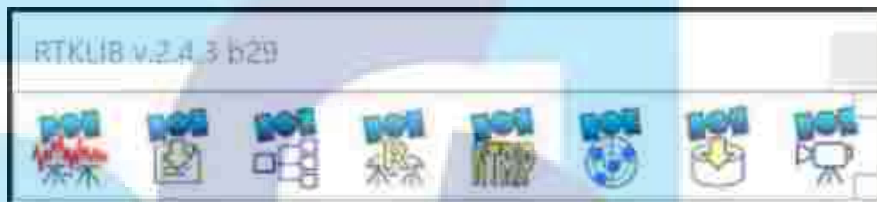


Figure 3.14 RTKLAUNCH

After observing all of the RTKLIB features, it is concluded that RTKNAVI and RTKPOST are the most suitable programs to be modified because RTKNAVI is useful for producing the RTK GNSS solution in real time and RTKPOST is useful for producing the RTK GNSS solution after post-processing, which can be useful for comparing experiment data.

3.2.3.2 Analyzing Unmodified RTKNAVI

As the first step to determine whether modifying the RTKNAVI is the appropriate action to do, the debug trace of RTKNAVI is analyzed. Debug trace of RTKNAVI contains all recorded information when the RTKNAVI is executed. After RTKNAVI is launched, in the options menu, the settings are adjusted as can be seen in the figures below. The positioning mode is set to kinematic, the frequencies chosen are L1, L2, and L5, and the elevation mask is set to be 30°. For the ease of analysis, the satellites constellations to be used are limited to GPS and QZSS. The other settings are left unchanged as default.

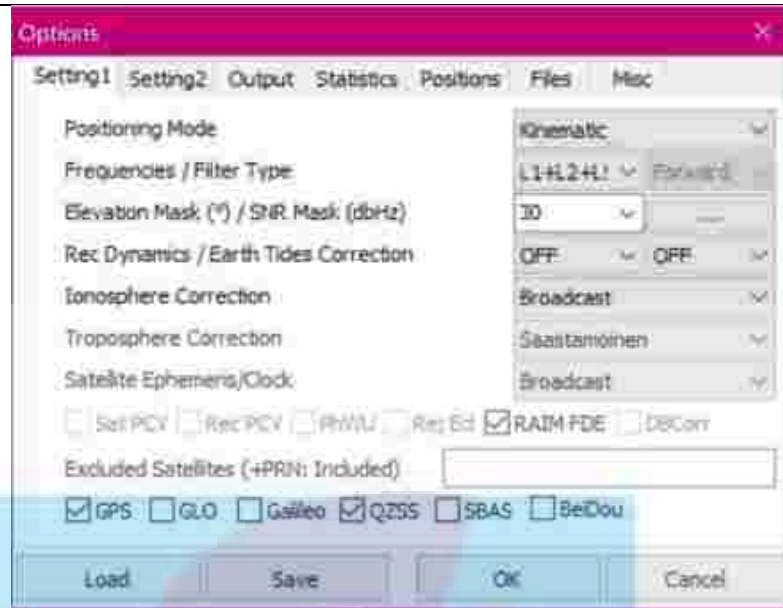


Figure 3.15 RTKNAVI Settings (1)

Integer ambiguity resolution is set to fix-and-hold for GPS. The other settings are left unchanged as default.



Figure 3.16 RTKNAVI Settings (2)

To enable the debug trace function, level 4 should be chosen in the output debug trace. The other settings are left unchanged as default.

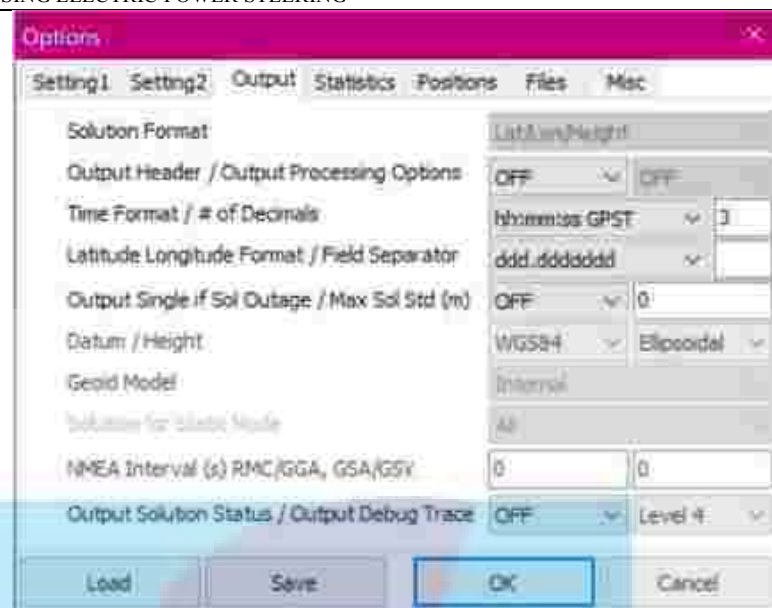


Figure 3.17 RTKNAVI Settings (3)

The base station coordinate is inputted based on latitude, longitude, and height obtained from averaging the single position for 30 minutes in ReachView. The other settings are remained unchanged as default.

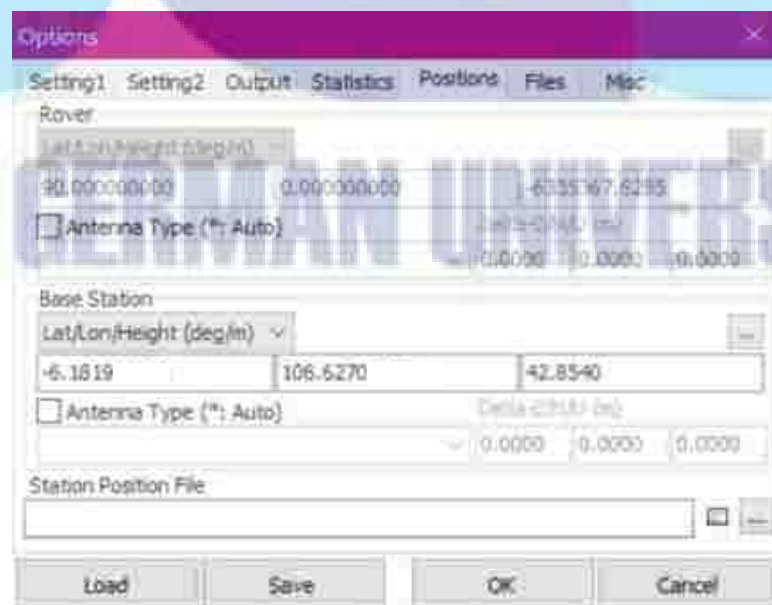


Figure 3.18 RTKNAVI Settings (4)

To stream base and rover data from NTRIP Caster, in RTKNAVI input menu, NTRIP Client is chosen with RTCM3 as the data type.



Figure 3.19 RTKNAVI Input Streams

In the option menu of rover input stream, the NTRIP Caster Host is filled with rtk2go.com since it is the name of the caster. The number of port used is 2101. The name of the mountpoint is RoverTest and the password is BETATEST.

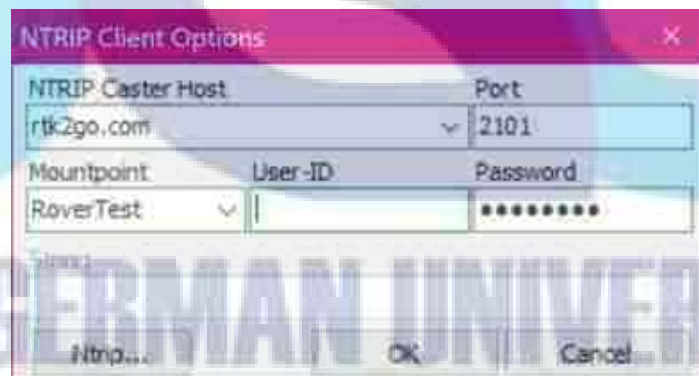


Figure 3.20 RTKNAVI Rover Input Stream

For the base input stream, the settings are the same with the rover input stream, since it uses the same NTRIP Caster. The only difference is the name of the mountpoint.

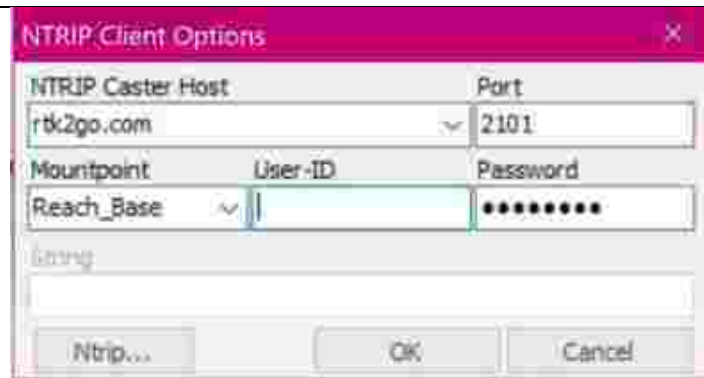


Figure 3.21 RTKNAVI Base Input Stream

After all of the RTKNAVI settings have been adjusted, the RTKNAVI is then executed for a few minutes to produce the debug trace data. After RTKNAVI debug trace data is obtained, it is compared with the rover and base station sky plots.

By observing the sky plots, it can be seen that in the rover station side, there are 14 visible satellites which are G01, G03, G06, G08, G09, G11, G17, G19, G22, G23, G28, J01, J02, and J03.



Figure 3.22 Rover Sky plot

However, in the base station side, there are 10 visible satellites which are G01, G03, G06, G09, G11, G17, G19, G22, G23, and G28.

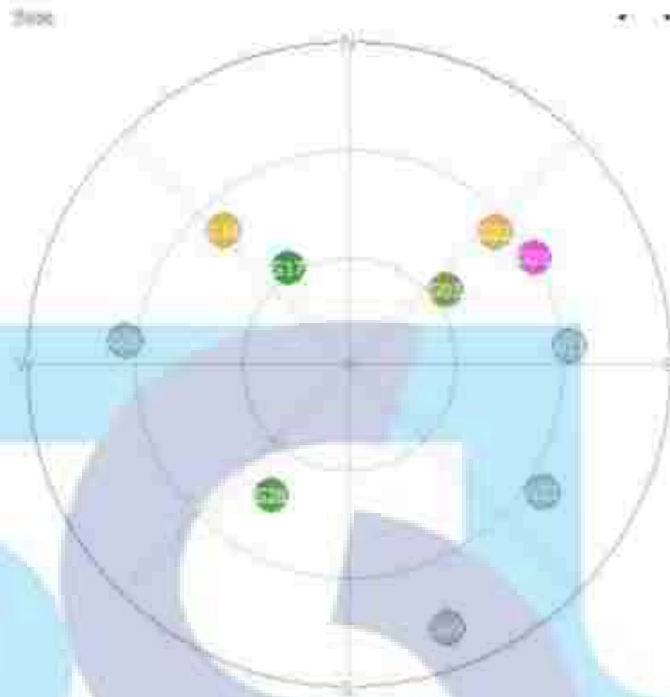


Figure 3.23 Base Sky plot

Some satellites are gray-colored because they aren't used to produce RTK solutions. There are a few possible reasons, which are, they might be broken satellites, they are excluded due to the elevation mask limitation (which is set to be 30° in this case), or they are only visible in either base or rover station side. For this case, satellites that are excluded due to elevation mask limitation are G06, G08, G09, G11, G23, and J02 for the rover station side and G06, G09, G11, and G23 for the base station side. As for J01 and J03, they are excluded from producing solution because they are only visible in the rover station side.

After analyzing the sky plots, RTKNAVI debug trace is observed. RTKNAVI debug trace contains the complete observation data for both base and rover station receiver. In the observation data, all of the visible satellites from base and rover station receiver are listed. "rcv1" refers to the rover station receiver and "rcv2" refers to the base station receiver.

Figure 3.24 Base and Rover Station Receiver Complete Observation Data

For the base station, the remaining satellites are G01, G03, G17, G19, G22, and G28.

Figure 3.25 Base Station Receiver Data After Masking

For the rover station, the remaining satellites are G01, G03, G17, G19, G22, G28, J01, and J03.

```
# sat= 1 azel= 80.2 20.1 elev=3447122.809 sig=3.048
# lonpccorr: time=2018/02/28 15:37:54.989 opt=1 sat= 3 pos=35.273 137.013 azel=53.182 55.782
# trppccorr: time=2018/02/28 15:37:54.989 opt=1 pos=35.273 137.013 azel=53.182 55.782
# sat= 3 azel= 53.2 55.8 elev=3447122.804 sig=3.631
# lonpccorr: time=2018/02/28 15:37:54.989 opt=1 sat=17 pos=35.273 137.013 azel=227.938 39.786
# trppccorr: time=2018/02/28 15:37:54.989 opt=1 pos=35.273 137.013 azel=227.938 39.786
# sat=17 azel=227.9 39.8 elev=3447122.809 sig=2.630
# lonpccorr: time=2018/02/28 15:37:54.989 opt=1 sat=19 pos=35.273 137.013 azel=317.102 39.831
# trppccorr: time=2018/02/28 15:37:54.989 opt=1 pos=35.273 137.013 azel=317.102 39.831
# sat=19 azel=317.1 39.8 elev=3447122.809 sig=3.739
# lonpccorr: time=2018/02/28 15:37:54.989 opt=1 sat=22 pos=35.273 137.013 azel=48.244 34.993
# trppccorr: time=2018/02/28 15:37:54.989 opt=1 pos=35.273 137.013 azel=48.244 34.993
# sat=22 azel= 48.2 35.0 elev=3447122.811 sig=3.782
# lonpccorr: time=2018/02/28 15:37:54.989 opt=1 sat=28 pos=35.273 137.013 azel=210.428 47.402
# trppccorr: time=2018/02/28 15:37:54.989 opt=1 pos=35.273 137.013 azel=210.428 47.402
# sat=28 azel=210.4 47.4 elev=3447122.809 sig=3.673
# lonpccorr: time=2018/02/28 15:37:54.989 opt=1 sat=19 pos=35.273 137.013 azel=317.028 34.269
# trppccorr: time=2018/02/28 15:37:54.989 opt=1 pos=35.273 137.013 azel=317.028 34.269
# sat=19 azel=317.0 34.3 elev=3447124.546 sig=3.348
# lonpccorr: time=2018/02/28 15:37:54.989 opt=1 sat=30 pos=35.273 137.013 azel=131.000 79.884
# trppccorr: time=2018/02/28 15:37:54.989 opt=1 pos=35.273 137.013 azel=131.000 79.884
# sat=30 azel=131.0 79.9 elev=3447122.809 sig=3.634
```

Figure 3.26 Rover Station Receiver Data After Masking

The common satellites between base and rover station receiver are also shown. These satellites are the only ones that will be used in producing RTK solutions.

```
4 ( 0) sat= 1 iu= 0 ir=14
4 ( 1) sat= 3 iu= 1 ir=15
4 ( 2) sat= 17 iu= 6 ir=19
4 ( 3) sat= 19 iu= 7 ir=20
4 ( 4) sat= 22 iu= 8 ir=21
4 ( 5) sat= 28 iu=10 ir=23
```

Figure 3.27 Common Satellites Between Base and Rover Station Receiver

After observing the RTKNAVI debug trace, it is concluded since RTKNAVI executes the satellites filtering based on elevation mask and satellites visibility, it is possible to modify the RTKNAVI by incorporating additional satellites filtering codes in the existing program.

3.2.3.3 Modifying RTKNAVI

The RTKNAVI modification is done in QT for Windows since the original program is already ported to QT. After analyzing the codes in RTKNAVI, it is found that in rtkpos.c, there is an algorithm to select common satellites between base and rover station receiver and also filter those satellites based on the elevation mask set by variable “elmin”. Therefore, it is concluded that the modification can be done in that particular section by incorporating the second elevation mask and heading direction value as an additional filter to select common satellites between base and rover station receiver.

```

/* select common satellites between rover and reference station */
static int select_sat(const obsd_t *obs, double azim1, int n1, int n2,
                    const posd_t *apnt, int azim2, int n3, int n4)
{
    int i, j, k, l;

    trace(3, "select_sat : n1=%d, n2=%d, n3=%d, n4=%d", n1, n2, n3, n4);

    for (i=0; i<n1; i++) {
        (obs[i].sat-obs[i].sat)++;
        if (obs[i].sat-obs[i].sat > 0) {
            if (obs[i].sat-obs[i].sat > 0) {
                if (azim1[i]>azim2-90) { /* elevation at base station */
                    sat[k]=obs[i].sat; k++;
                    trace(4, "Sat: %d, azim: %d, elev: %d", k, obs[i].sat, i);
                }
            }
        }
    }
    return k;
}

```

Figure 3.28 Original rtkpos.c

The first step is to add additional QComboBox in the navipt.ui for the second elevation mask box. The original elevation mask box named “ElMask” will be used for masking the satellites in the heading direction and the second elevation mask box named “ElMask2” will be used for masking the satellites in the perpendicular heading direction.

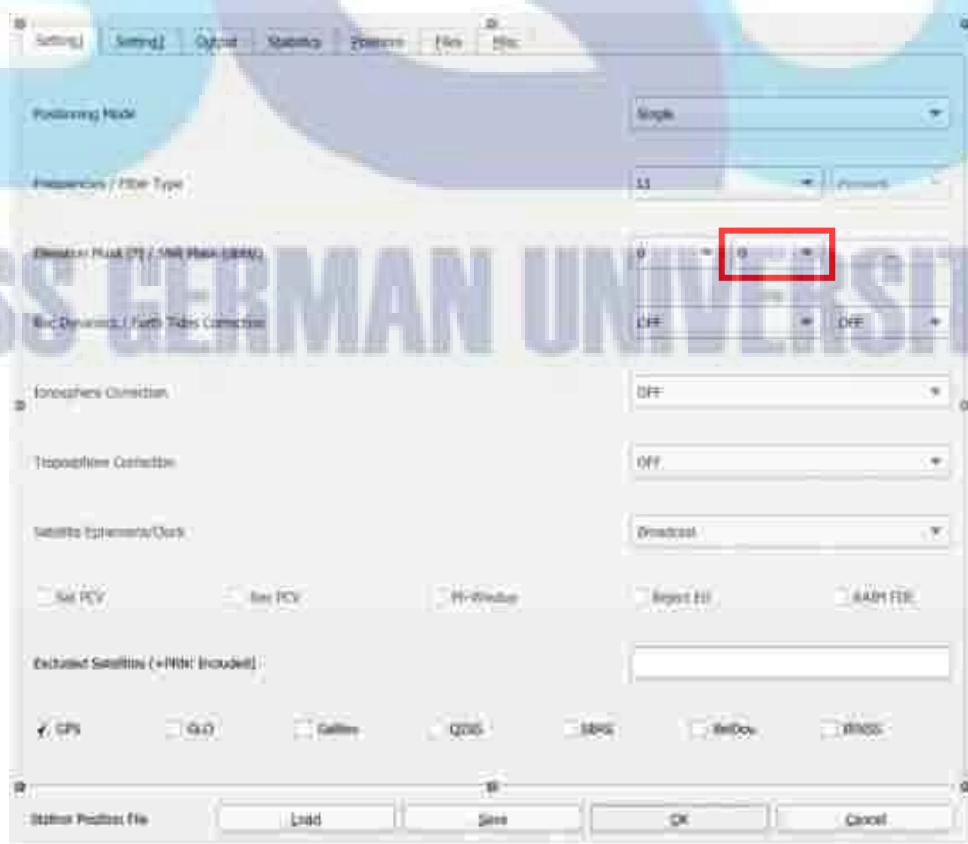


Figure 3.29 Modified navipt.ui

After that, additional QLineEdit named “HeadingDir” is added in the navimain.ui for inputting the heading direction value.

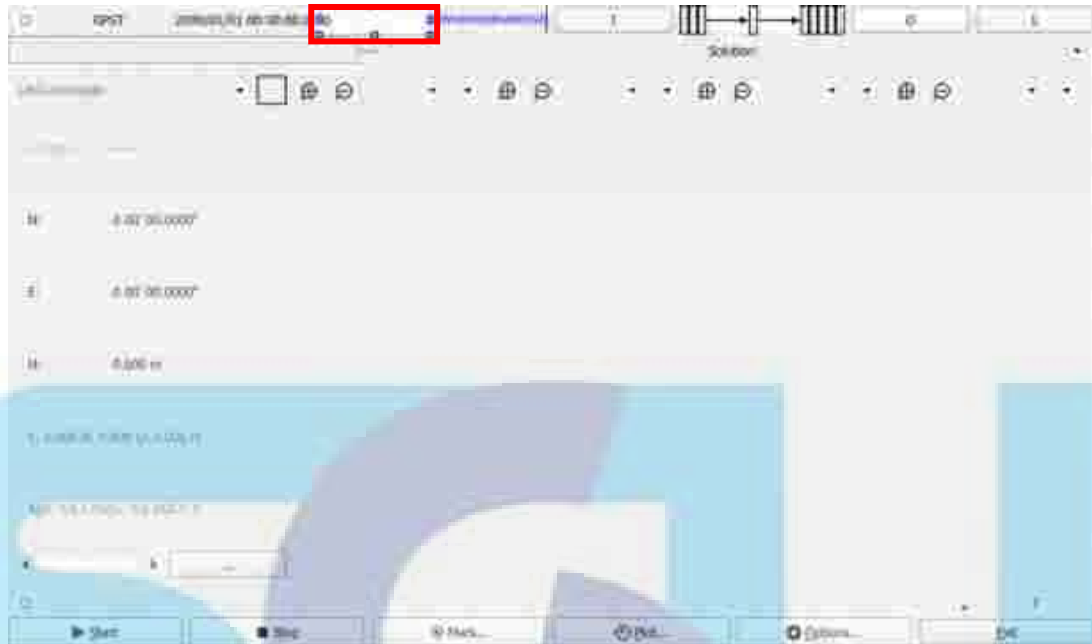


Figure 3.30 Modified navimain.ui

In rtklib.h, the variables for the second elevation mask and heading direction are defined. In this case, they are named “elmin2” and “heading”. Both are members under a structure named `prcopt_t` (a valid identifier for the members inside the curly brackets). The `prcopt_t` structure is the same structure as the first elevation mask variable named “elmin”.

```
typedef struct {
    int mode; /* positioning mode (PHODE_???) */
    int soltype; /* solution type (2:forward,1:backward,3:combined) */
    int nf; /* number of frequencies (3:1,3:1+15,3:1+1+15) */
    int navsys; /* navigation system */
    double elmin; /* elevation mask angle (rad) */
    double elmin2; /* elevation mask angle (rad) */
    double heading; /* heading direction (degree) */
    double srmask; /* SRM mask */
    int satopt; /* satellite option (EPHOP_???) */
    int modear; /* AR mode (0:off,1:continuous,2:instantaneous,3:fix and hold,4:stop-ar) */
    int glomaskar; /* GLASS AR mode (0:off,1:on,2:auto cal,3:ext cal) */
    int bdsmaskar; /* BeiDou AR mode (0:off,1:on) */
    int maxcorr; /* max range count to reset fix */
    int withlock; /* with lock count to fix ambiguity */
    int withfix; /* with fix count to hold ambiguity */
    int maxiter; /* max iteration to resolve ambiguity */
    int lonopt; /* longitude option (LONOPT_???) */
    int tropopt; /* troposphere option (TROPOPT_???) */
    int dynamics; /* dynamics model (0:pos,1:velocity,2:accel) */
    int tidecorr; /* earth tide correction (0:off,1:solid,2:solid+ocean+pole) */
    int iter; /* number of filter iteration */
    int codesmooth; /* code smoothing window size (frames) */
    int intpref; /* interpolate reference obs (for post mission) */
    int sbascorr; /* SBAS correction options */
    int sbasatsel; /* SBAS satellite selection (sall) */
    int rovpas; /* rover position for fixed mode */
    int refpos; /* base position for relative mode */
    /* (0:pos in arcorr, 1:average of single pos, */
    /* 2:read from file, 3:fix header, 4:fix obs) */
    double aratio(NFREQ); /* code/phase error ratio */
    double err[5]; /* measurement error factor */
    /* 0:reserved */
    /* 1-3: error factor of phase (m) */
    /* 4: doppler frequency (hz) */
    double std[3]; /* initial state std (0:fix,1:iono [2]trop) */
    double prn[6]; /* process noise std (0:fix,1:iono [2]trop [3]accel [4]accv [N] pos) */
    double sclkstab; /* satellite clock stability (1/sec) */
    double thresar[N]; /* AR validation threshold */
    double elmaskar; /* elevation mask of AR for rising satellite (deg) */
    double elmaskhold; /* elevation mask to hold ambiguity (deg) */
    double thresalp; /* slip threshold of geometry-free phase (m) */
    double maxdiff; /* max difference of time (sec) */
    double maxiono; /* reject threshold of ionization (m) */
    double maxdop; /* reject threshold of dop (m) */
    double baselen[2]; /* baseline length constraint (convet,signs) (m) */
    double ru[2]; /* rover position for fixed mode (x,y,z) (m) */
    double rb[2]; /* base position for relative mode (x,y,z) (m) */
    char anttype[2][MAXANT]; /* antenna types (rover,base) */
    double antdel[2][3]; /* antenna delta (x,y,z) (m) (ref_x,ref_y,ref_z) */
    double pcvr[2]; /* receiver antenna parameters (rover,base) */
    unsigned char exsats[MAXSAT]; /* excluding satellites (1:excluded,2:included) */
    int maxavep; /* max averaging epoches */
    int initstat; /* initialize by restart */
    int outsingle; /* output single by apps/float/fix/app outage */
    char rsopt[2][256]; /* rfix options (rover,base) */
    int posopt[8]; /* positioning options */
    int syncsol; /* solution sync mode (0:off,1:on) */
    double oclap[2][6*11]; /* ocean tide loading parameters (x,y,base) */
    extern t_gstex; /* extended receiver error model */
    int freqopt; /* disable L2-AR */
    char puopt[256]; /* pps option */
} prcopt_t;
```

Figure 3.31 Modified Structure prcopt_t

In navimain.cpp, there is a function named TimerTimer that is executed every 100ms. Thus, so that the heading value is executed every 100ms as well, the heading variable is considered to be put inside the TimerTimer function.

```
connect(&Timer, SIGNAL(timeout()), this, SLOT(TimerTimer()));  
Timer.setInterval(100);  
Timer.setSingleShot(false);  
Timer.start();
```

Figure 3.32 TimerTimer Function Update Time

In the TimerTimer function itself, the rtksvr variable is always called for server lock and server unlock. Therefore, the heading variable is considered to be passed through rtksvr, because it is always executed.

```
void MainWindow::TimerTimer()  
{  
    static int n = 0, inactive = 0;  
    sol_t *sol;  
    int i, update = 0;  
  
    trace(4, "TimerTimer()\n");  
    rtksvrlock(&rtksvr);  
  
    for (i = 0; i < rtksvr.nsol; i++) {  
        sol = rtksvr.solbuf + i;  
        UpdateLog(sol->stat, sol->time, sol->rr, sol->qr, rtksvr.rtk.rb, sol->ns,  
            sol->age, sol->ratio);  
        update = 1;  
    }  
    rtksvr.nsol = 0;  
    solCurrentStat = rtksvr.state ? rtksvr.rtk.sol.stat : 0;  
    rtksvrunlock(&rtksvr);  
}
```

Figure 3.33 TimerTimer Function - Server Lock and Server Unlock

The rtksvr variable is assigned under rtksvr_t structure.

```
extern rtksvr_t rtksvr; // rtk server struct  
extern stream_t monistr; // monitor stream
```

Figure 3.34 rtksvr Variable

After observing rtksvr_t structure, it is known that variable rtk is assigned under rtk_t structure.

```
typedef struct {
    /* RTK server type */
    int state; /* server state (0:stop,1:running) */
    int cycle; /* processing cycle (ms) */
    int rmcycle; /* NMEA request cycle (ms) (0:nn req) */
    int rmcrrq; /* NMEA request (0:np,1:imrppu,2:single sol) */
    double rmcrrpos[3]; /* NMEA request position (ecef) (m) */
    int bufsize; /* input buffer size (bytes) */
    int format[3]; /* input format (rov,base,corr) */
    solopt_t solopt[2]; /* output solution options (sol1,sol2) */
    int navesel; /* ephemeris select (0:all,1:rover,2:base,3:corr) */
    int nsbs; /* number of sbas message */
    int nsnt; /* number of solution buffer */
    rtk_t rtk; /* RTK control/result struct */
    int nb[3]; /* bytes in input buffers (rov,base) */
    int nsb[3]; /* bytes in solution buffers */
    int npb[3]; /* bytes in input peak buffers */
    unsigned char *buff[3]; /* input buffers (rov,base,corr) */
    unsigned char *sbuff[3]; /* output buffers (sol1,sol2) */
    unsigned char *pbuff[3]; /* peak buffers (rov,base,corr) */
    sol_t solbuf[MAXSOLBUF]; /* solution buffer */
    unsigned int rmcmsg[3][10]; /* input message counts */
    raw_t raw[3]; /* receiver raw control (rov,base,corr) */
    rtcn_t rtcn[3]; /* RTCN control (rov,base,corr) */
    gtime_t ftime[3]; /* download time (rov,base,corr) */
    char files[3][MAXSTPATH]; /* download paths (rov,base,corr) */
    obs_t obs[3][MAXOBSBUF]; /* observation data (rov,base,corr) */
    nav_t nav; /* navigation data */
    sbasmsg_t sbasmsg[MAXSBMSG]; /* SBAS message buffer */
    stream_t stream[8]; /* streams (rov,base,corr,sol1,sol2,logr,logp,logs) */
    stream_t stream; /* monitor stream */
    unsigned int tick; /* start tick */
    thread_t thread; /* server thread */
    int cputime; /* CPU time (ms) for a processing cycle */
    int precount; /* missing observation data count */
    int naves; /* number of averaging base pos */
    double rb_ave[3]; /* averaging base pos */
    char cmds_periodic[1][MAXRCVCMDB]; /* periodic commands */
    char cmd_reset[MAXRCVCMDB]; /* reset command */
    double bl_reset; /* baseline length to reset (km) */
    lock_t lock; /* lock flag */
} rtksvr_t;
```

Figure 3.35 rtksvr_t Structure

Then, after observing rtk_t structure, it is also known that variable opt is assigned under precount_t structure.

```
typedef struct {
    /* RTK control/result type */
    sol_t sol; /* RTK solution */
    double rb[6]; /* base position/velocity (ecef) (m/m/s) */
    int nx,nv; /* number of float states/fixed states */
    double tt; /* time difference between current and previous (s) */
    double *x,*P; /* float states and their covariance */
    double *xs,*Ps; /* fixed states and their covariance */
    int nfix; /* number of continuous fixes of ambiguity */
    ambc_t ambc[MAXSAT]; /* ambiguity control */
    spat_t spat[MAXSAT]; /* satellite status */
    int neb; /* bytes in error message buffer */
    char errbuf[MAXERRMSG]; /* error message buffer */
    precount_t opt; /* processing options */
} rtk_t;
```

Figure 3.36 rtk_t Structure

Therefore, as a conclusion, since all of the variables are already linked, the heading variable can be passed from the variable opt to rtk and finally to rtksvr. In the TimerTimer function, the passing parameter for heading is added. Because the QLineEdit read the values in it as text but the heading variable is defined as double, a conversion from text to double must be done.

```

void MainWindow::TimerTimer()
{
    static int n = 0, inactive = 0;
    sol_t *sol;
    int i, update = 0;

    trace(4, "TimerTimer()");

    rtksvr.rtk.opt.heading=HeadingDir->text().toDouble();

    rtksvrlock(&rtksvr);

    for (i = 0; i < rtksvr.ncol; i++) {
        sol = rtksvr.solbuf + i;
        UpdateSol(sol->stat, sol->time, sol->err, sol->qr, rtksvr.rtk.cb, sol->hw,
                sol->age, sol->ratio);
        update = 1;
    }
    rtksvr.ncol = 0;
    solCurrentStat = rtksvr.state ? rtksvr.rtk.sol.stat : 0;
    rtksvrunlock(&rtksvr);
}

```

Figure 3.37 Modified TimerTimer Function in navimain.cpp

The default processing option for the second elevation mask and heading direction in the rtkcmn.c must be defined as well. For the second elevation mask, the default is set as 15°, which is the same as the first elevation mask. For the heading direction, it is set as 0°.

```

static prcopt_t prcopt_default = { /* defaults processing options */
    PMODE_SINGLE, 0.2, SYS_GPS, /* mode, soltype, nf, nvarsys */
    15.0*B2R, 15.0*B2R, 0.00, {{0,0}}, /* elmin, elmin2, heading, thrmask */
    0, 1, 1, 1, /* satph, codevar, glomdevar, hdsmdvar */
    5, 0, 10, 1, /* maxout, minlock, minfix, amaxkiter */
    0, 0, 0, 0, /* estion, eattrap, dynamics, tidecorr */
    1, 0, 0, 0, 0, /* iter, codesmooth, intpref, abscorr, shhssatsel */
    0, 0, /* covpos, refpos */
    {100.0, 100.0}, /* e(atio) */
    {100.0, 0.003, 0.003, 0.0, 1.0}, /* ar[] */
    {10.0, 0.03, 0.3}, /* std[] */
    {1E-4, 1E-3, 1E-4, 1E-1, 1E-2, 0.0}, /* ppp[] */
    5E-12, /* ac[kstabil] */
    {3.0, 0.0999, 0.25, 0.1, 0.05}, /* thressar */
    0.0, 0.0, 0.05, /* simaxvar, almaskhold, thresslip */
    30.0, 30.0, 30.0, /* maxtdif, maxiono, maxgdp */
    {0}, {0}, {0}, /* baseline, cu, cb */
    {""}, /* anttype */
    {{0}}, {{0}}, {0} /* antdel, pcv, ssasats */
};

```

Figure 3.38 Modified rtkcmn.c

The function GetOpt is modified by adding the code to set the index of the “ElMask2” with the index returned from QComboBox “ElMask2”. The index type is text.

```
void OptDialog::GetOpt(void)
{
    QLineEdit *edit0[] = { RsvPos1, RsvPos2, RsvPos3 };
    QLineEdit *edit1[] = { RsvPos1, RsvPos2, RsvPos3 };

    PosMode->setCurrentIndex(PrcOpt.mode);
    Freq->setCurrentIndex(PrcOpt.nf - 1 * NFREQ - 1 * NFREQ - 1 * PrcOpt.nf - 1);
    ElMask->setCurrentIndex(ElMask2->currentIndex());
    ElMask2->setCurrentIndex(ElMask2->findText(QString::number(PrcOpt.elmin2 * RD)));
}
```

Figure 3.39 Modified GetOpt Function in naviopt.cpp

The function SetOpt is modified by adding the code to convert the value of QComboBox “ElMask2” from text to double and assign the converted value to variable “elmin2”. The variable elmin2 is a member under prcopt_t structure and its variable PrcOpt.

```
public:
    OptDialog *optDialog;
    QString InitFile;

    int PanelStack, PanelMode;
    int SvcCycle, SvcBuffSize, Scale, SolBuffSize, NavSelect, SavedSol;

    int NavSelEq, NavSelCycle, InTimeTag, InTime64Bit;
    int OutTimeTag, OutAppend, LogTimeTag, LogAppend;
    int TimeOutTime, ReconTime, SbsCorr, DgpsCorr, TideCorr, FileSwapMargin;
    int StrLen[MAXSTRLEN], StrLenC[MAXSTRLEN], Format[MAXSTRLEN];
    int CmdEna[3][3], CmdEnaTpc[3][3];
    int TimeSys, SolType;
    int PlotType[4], FreqType[4];
    int TrkType[4];
    int TrkScale[4];
    int RvMode[4];
    int MonPort, OpenPort;

    int Pos1, Pos2, Pos3, Rsv[2], SolCurrentStat;
    int Set[2][MAXSAT], Shr[2][MAXSAT][NFREQ], Vset[2][MAXSAT];
    double Az[2][MAXSAT], El[2][MAXSAT];
    gtime_t *time;
    int *SolStat, *NavStat;
    double *SolRsv, *SolRef, *Qr, *VelHuv, *Age, *Rvdx;
    double TrkR[2], NavR[2];

    QString Paths[MAXSTRLEN][4], Cmds[3][3], CmdsTpc[3][3];
    QString InTimeStart, InTimeSpeed, ExSat;
    QString NavOpt[3], ProxyAddr;
    QString OutSwapInterval, LogSwapInterval, ResetCmd;

    prcopt_t PrcOpt;
```

Figure 3.40 PrcOpt Variable

```
void OptDialog::SetOpt(void)
{
    QString FieldSep_Text = FieldSep->text();
    QLineEdit *editu[] = { RowPos1, RowPos2, RowPos3 };
    QLineEdit *editr[] = { RefPos1, RefPos2, RefPos3 };

    PrcOpt.mode = PseMode->currentIndex();
    PrcOpt.inf = Freq->currentIndex() * 2;
    PrcOpt.elmin1 = ElMask->currentText().toDouble() * D2R;
    PrcOpt.elmin2 = ElMask2->currentText().toDouble() * D2R;
}
```

Figure 3.41 Modified SetOpt function in naviopt.cpp

The function LoadOpt is modified by adding the code to set the value of “elmin2” with the value obtained from the initialization file. The initialization file contains the last value and settings from the last configuration before RTKNAVI is closed.

```
void MainForm::loadOpt(void)
{
    QSettings settings(INIFile, QSettings::IniFormat);
    QString sz;
    QStringList strdel = { 0, 1, 2, 3, 4, 5, 7, 8 };
    QString szPath;

    for (i = 0; i < 3; i++) {
        sz = strdel[i];
        Stream[i] = settings.value(QString("stream/stream%1").arg(sz), 0).toInt();
        Stream[i] = settings.value(QString("stream/stream%1").arg(sz), 0).toInt();
        Format[i] = settings.value(QString("stream/format%1").arg(sz), 0).toInt();
        for (j = 0; j < 4; j++)
            Path[i][j] = settings.value(QString("stream/path_%1_%2").arg(sz).arg(j), "").toString();
    }

    for (i = 0; i < 3; i++)
        RowOpt[i] = settings.value(QString("stream/rowopt%1").arg(i + 1), "").toString();
    for (i = 0; i < 3; i++) for (j = 0; j < 3; j++) {
        Code[i][j] = settings.value(QString("serial/code_%1_%2").arg(i).arg(j), "").toString();
        Code[i][j] = settings.value(QString("serial/code_%1_%2").arg(i).arg(j), 0).toInt();
        Code[i][j].replace("00", "10");
    }

    for (i = 0; i < 3; i++) for (j = 0; j < 3; j++) {
        Code[i][j] = settings.value(QString("code_%1_%2").arg(i).arg(j), "").toString();
        Code[i][j] = settings.value(QString("code_%1_%2").arg(i).arg(j), 0).toInt();
        Code[i][j].replace("00", "10");
    }

    PrcOpt.mode = settings.value("prcopt/mode", 0).toInt();
    PrcOpt.inf = settings.value("prcopt/inf", 2).toInt();
    PrcOpt.elmin1 = settings.value("prcopt/elmin1", 0.0).toDouble();
    PrcOpt.elmin2 = settings.value("prcopt/elmin2", 0.0 * D2R).toDouble();
}
```

Figure 3.42 Modified LoadOpt Function in navimain.cpp

The function SaveOpt is modified by adding the code to save the last value used of “elmin2” to the initialization file. The initialization file is saved automatically when the RTKNAVI is closed.

```
void Navimain::SaveOpt(void)
{
    QSettings settings(InFile, QSettings::IniFormat);
    int i, j, no; charno[4] = { 0, 1, 0, 2, 3, 4, 5, 6 };

    trcs[2] = 7344000000;

    for (i = 0; i < 4; i++) {
        no = strno[i];
        settings.setValue(QString("stream/stream%1").arg(no), Stream[i]);
        settings.setValue(QString("stream/stream%1").arg(no), Stream[i]);
        settings.setValue(QString("stream/format%1").arg(no), Format[i]);
        for (j = 0; j < 4; j++)
            settings.setValue(QString("stream/path_%1_%2").arg(no).arg(j), PathH[j][i]);
    }

    for (i = 0; i < 3; i++)
        settings.setValue(QString("stream/corrupt%1").arg(i + 1), RcvOpt[i]);
    for (i = 0; i < 3; i++) for (j = 0; j < 3; j++) {
        Cmds[i][j].replace("\\r\\n", "\\n");
        settings.setValue(QString("scpt/cmd_%1_%2").arg(i).arg(j), Cmds[i][j]);
        settings.setValue(QString("scpt/cmdena_%1_%2").arg(i).arg(j), CmdEne[i][j]);
    }

    for (i = 0; i < 2; i++) for (j = 0; j < 2; j++) {
        CmdsTep[i][j].replace("\\r\\n", "\\n");
        settings.setValue(QString("scpt/cmd_%1_%2").arg(i).arg(j), CmdsTep[i][j]);
        settings.setValue(QString("scpt/cmdena_%1_%2").arg(i).arg(j), CmdEneTep[i][j]);
    }

    settings.setValue("prcopt/enmode", PrcOpt.enmode);
    settings.setValue("prcopt/nf", PrcOpt.nf);
    settings.setValue("prcopt/elmin", PrcOpt.elmin);
    settings.setValue("prcopt/elmin2", PrcOpt.elmin2);
}
```

Figure 3.43 Modified SaveOpt Function in navimain.cpp

After the variables are already declared, the codes in rtkpos.c are modified. First, the variables are declared, including the heading variable that is taken from variable opt that is assigned under prcopt_t structure. Then, the difference between each satellite's azimuth and heading direction is calculated. The result is stored in variable "selisih". If the result is a negative number, then it is added by 360, so that the result will always be positive. The first step for filtering the satellites is to include only the common satellites between base and rover receiver to create RTK solutions. The second step is to include only the common satellites that is ranging from -30° until $+30^{\circ}$ and $+150^{\circ}$ until $+210^{\circ}$ with the heading direction, which has elevation angle greater than "elmin" variable. The third step is to include the rest of the common satellites which have elevation angle greater than "elmin2" variable.

```

/* select common satellites between rover and reference station */
static int select(const obsd_t *obs, const azel_t *azel, int nu, int nr,
               const prcpt_t *prcpt, int *stat, int *lu, int *le)
{
    int i, j, k=0;
    double selstat[1000];
    double heading=prcpt->heading;

    trace(3, "select: i: %d\n", nu, nr);

    for (i=0; i<nu; i++) {
        for (j=0; j<nr; j++) {
            selstat[j*2] = (azel[j*2]*R2D - heading);
            if (selstat[j*2] < 0)
                selstat[j*2] += 360;
        }
        if (obs[i].sat < obs[j].sat) i--;
        else if (obs[i].sat > obs[j].sat) i++;
    }

    if ((selstat[i*2] > 330 || selstat[i*2] < 30 || (selstat[i*2] > 150 && selstat[i*2] < 210))
        || (azel[i*2] > prcpt->heading))
    {
        sat[k]=obs[i].sat; lu[k]=i; le[k++]=j;
        selstat[j*2] = (azel[j*2]*R2D - heading);
    }

    else if ((selstat[i*2] < 330 && selstat[i*2] > 210) || (selstat[i*2] > 30 && selstat[i*2] < 150))
    {
        if (azel[i*2] > prcpt->heading)
        {
            sat[k]=obs[i].sat; lu[k]=i; le[k++]=j;
            selstat[j*2] = (azel[j*2]*R2D - heading);
        }
    }
}

return k;

```

Figure 3.44 Modified rtkpos.c

Additional codes are added in the DrawSky function to improve the sky plot visualization in RTKNAVI. The additional codes include adding yellow tint for the area that has been masked by the first and second elevation masks, which are “elmin” and “elmin2”. This means that only the satellites that appear in the yellow area are included for creating RTK solutions. In addition to that, a sign to determine the heading direction is also added.

Considering the fact that every time a new canvas is made, the new canvas will be stacked on top of the older canvas, the modification is made as follows. First, a blank white canvas is made. In that particular canvas, the outer most ellipse for 0° elevation mask is drawn. After that, a yellow area is drawn first before the ellipses for 30° and 60° so that the other ellipses will not be hidden under the yellow area. Due to the fact that the elevation mask in the perpendicular heading direction will be smaller than the one in the heading direction, another ellipse with yellow filling is made after the variable

“elmin2”. Then, a pie with yellow filling is made after the variable “elmin”. After the yellow area is finished, the heading sign is made by using chord. Finally, the ellipses for 30° and 60° are made.

```
void MainWindow::DrawSky(QPainter *c, int w, int h, int x0, int y0)
{
    QString label[] = { tr("N"), tr("E"), tr("S"), tr("W") };

    //middle point of the canvas. x0, y0 is located in the most upper left side of the monitor.
    QPoint p(x0 + w / 2, y0 + h / 2);

    //r is the radius. Only 95% of the minimum length&width of the canvas is allowed to be use.
    double r = MIN(w * 0.95, h * 0.95) / 2;
    int x, y, d, x1, y1;

    //outer most ellipse
    c->setBrush(Qt::white);
    c->drawEllipse(p.x()-r, p.y()-r, 2*r+1, 2*r+1);

    //draw the yellow ellipse for "elmin2"
    d = static_cast<int>(r * (90 - PrcOpt.elmin2*R2D) / 90);
    c->setBrush(Qt::yellow);
    c->setPen(Qt::NoPen);
    c->drawEllipse(p.x()-d+1, p.y()-d, 2*d, 2*d);

    //draw the yellow pie for "elmin"
    d = static_cast<int>(r * (90 - PrcOpt.elmin*R2D) / 90);
    double heading = (60-HeadingDir->text().toDouble())*16;
    c->drawPie(p.x()-d+1, p.y()-d, 2*d, 2*d, heading, 60*16);
    c->drawPie(p.x()-d+1, p.y()-d, 2*d, 2*d, heading+(180*16), 60*16);

    //draw heading direction with chord
    c->setPen(Qt::red);
    c->drawChord(p.x()-d+1, p.y()-d, 2*d, 2*d, heading, 60*16);

    //second outer most & third outer most ellipse
    c->setBrush(Qt::NoBrush);
    c->setPen(Qt::NoPen);
    for (e = 30; e < 90; e += 30) {
        d = static_cast<int>(r * (90 - e) / 90);
        c->setPen(e == 30 ? Qt::gray : QColor(0xc0, 0xc0, 0xc0));
        c->drawEllipse(p.x()-d, p.y()-d, 2*d+1, 2*d+1);
    }

    for (a = 0; a < 360; a += 45) {
        x = static_cast<int>(r * sin(a * D2R));
        y = static_cast<int>(r * cos(a * D2R));
        c->setPen(QColor(0xc0, 0xc0, 0xc0));
        c->drawLine(p.x(), p.y(), p.x()+x, p.y()-y);
        if (a % 90 == 0) DrawText(c, p.x() + x, p.y() - y, label[a / 90], Qt::gray, 1);
    }
}
```

Figure 3.45 Modified DrawSky Function in navimain.cpp

Due to the initial plan that the heading direction value will be received via TCP, it becomes necessary to implement the TCP communication function in RTKNAVI. In this thesis, the TCP codes were taken from an open source site and implemented accordingly by adding or eliminating some elements from the original codes.

The TCP file codes which are serverStuff.h and serverStuff.cpp is copied physically to the appcmn_qt folder. The header file can be found under the Headers folder.



Figure 3.46 Adding serverStuff.h

The source file can be found under the Sources folder.



Figure 3.47 Adding serverStuff.cpp

To enable TCP connection function, the network module must be added in the list of QT modules that is being used in RTKNAVI project. This can be found in the pro file of RTKNAVI.

```
QT += widgets core gui serialport network
```

Figure 3.48 Modified rtknavi_qt.pro

To be able to use the TCP codes that has been copied, the header file of serverStuff must be included in the navimain.h.

```
#include <QDialog>
#include <QColor>
#include <QSystemTrayIcon>
#include <QTimer>
#include <QMenu>

#include "rtklib.h"
#include "rtk_navimain.h"
#include "serverStuff.h"
```

Figure 3.49 Modified navimain.h (1)

Later, there will be an additional function named gotNewMessage in navimain.cpp, therefore, it must be first defined in the public slots of navimain.h.

```
public slots:
    void TimerTimer();

    void BtnStartClick();
    void BtnStopClick();
    void gotNewMessage(QString msg);
    void BtnPlotClick();
    void BtnOptClick();
    void BtnExitClick();
```

Figure 3.50 Modified navimain.h (2)

In the private class members, the pointer for ServerStuff is defined.

```
private:
    FILE * TIEData;
    ServerStuff *server;
    AboutDialog *aboutDialog;
    InputStrDialog *inputStrDialog;
    OutputStrDialog *outputStrDialog;
    LogStrDialog *logStrDialog;
    MonitorDialog *monitor;
    QSystemTrayIcon *systemTray;
    QMenu *trayMenu;
    QAction *MenuStartAction, *MenuStopAction, *MenuExitAction;
```

Figure 3.51 Modified navimain.h (3)

In the constructor, additional codes for TCP connection establishment is added.

```
// constructor
MainWindow::MainWindow(QWidget *parent)
: QDialog(parent)
{
    mainForm = Ui::ui;
    setupUi(this);

    server = new ServerStuff(this);
    connect(server, &ServerStuff::gotNewMessage,
            this, &MainWindow::gotNewMessage);
}
```

Figure 3.52 Modified Constructor in navimain.cpp

The BtnStartClick function is modified so that when the start button is pressed, the TCP connection is also started.

```
void MainWindow::BtnStartClick()
{
    trace(3, "BtnStartClick\n");

    if (!server->tcpServer->listen(QHostAddress::Any, 4800))
    {
        trace(3, "The port is taken by some other service\n");
        return;
    }
    connect(server->tcpServer, &TcpServer::newConnection, server, &ServerStuff::newConnection);
    trace(3, "Server started, port is opened.\n");

    SysStart();
}
```

Figure 3.53 Modified BtnStartClick Function in navimain.cpp

The BtnStopClick function is modified so that when the stop button is pressed, the TCP connection is also stopped.

```
void MainWindow::BtnStopClick()
{
    trace(3, "BtnStopClick\n");

    if (server->tcpServer->isListening())
    {
        disconnect(server->tcpServer, &TcpServer::newConnection, server, &ServerStuff::newConnection);

        QList<QTcpSocket> & clients = server->getClients();
        for (int i = 0; i < clients.count(); i++)
        {
            //server->sendToClient(clients.at(i), "Connection closed");
            server->removeClient(clients.at(i), "b");
        }

        server->stopServer->close();
        trace(3, "Server stopped, port is closed\n");
    }
    else
    {
        trace(3, "Error! Server was not running\n");
    }

    SysStop();
}
```

Figure 3.54 Modified BtnStopClick Function in navimain.cpp

Additional function named `gotNewMessage` is added so that whenever it receives the heading value, it will pass the value to `QLineEdit` “HeadingDir” and the heading value will be shown in the user interface.

```
void MainWindow::gotNewMessage(QString msg)
{
    HeadingDir->setText(msg);
}
```

Figure 3.55 Adding gotNewMessage Function in navimain.cpp

The `ServerStuff` function in `serverStuff.cpp` is modified because the original version can only work if it is communicating with its TCP client pair made by the programmer. In addition, the new TCP server is now able to distinguish whether the heading data that is received is valid or not by a certain criterion. In this case, the heading data is considered to be valid only if the number of character received equal to 4, there is only one character “h”, and it is located at the beginning of the data that has been received. If the heading data is valid, then the numbers after character “h” will be used as the heading direction value.

```
void ServerStuff::readClient()
{
    int dataValid=0;
    QTcpSocket *clientSocket = (QTcpSocket*)sender();

    QString rawData = clientSocket->readAll();
    qDebug() << "rawdata=" << rawData.toUtf8();

    if (rawData.startsWith("h", Qt::CaseSensitive)
        && rawData.count(QChar('h'), Qt::CaseSensitive)==1
        && rawData.count()==4)
    {
        dataValid=1;
        qDebug() << "Data valid";
    }
    else
    {
        dataValid=0;
        qDebug() << "Data invalid";
    }

    if (dataValid==1)
    {
        QString data = rawData.remove(QChar('h'), Qt::CaseSensitive);
        qDebug() << "data=" << data.toUtf8();
        emit gotNewMessage(data);

        if (sendToClient(clientSocket, QString("Reply: received [%1]").arg(data)) == -1)
        {
            qDebug() << "Some error occurred";
        }
    }
}
```

Figure 3.56 Modified ServerStuff Function in serverStuff.cpp

The modification of RTKPOST is done in QT for Windows as well. An additional QComboBox and QLineEdit are added in the postopt.ui for the second elevation mask box and heading direction value.



```
SWISS GERMAN UNIVERSITY
```

Figure 3.58 Modified postmain.h

The function GetOpt is modified by adding the code to set the value of “ElMask2” with the value returned from QComboBox “ElMask2”. In the other hand, the “HeadingDir” value is set with the number returned from QLineEdit “HeadingDir”. Both variables are in text type and configured so that they display only the significant figures.

```
void OptDialog::GetOpt(void)
{
    QLineEdit *editu[] = { RovPos1, RovPos2, RovPos3 };
    QLineEdit *editr[] = { RefPos1, RefPos2, RefPos3 };

    PosMode->setCurrentIndex(mainForm->PosMode);
    Freq->setCurrentIndex(mainForm->Freq);
    Solution->setCurrentIndex(mainForm->Solution);
    ElMask->setCurrentText(QString::number(mainForm->ElMask, 'f', 0));
    ElMask2->setCurrentText(QString::number(mainForm->ElMask2, 'f', 0));
    HeadingDir->setText(QString::number(mainForm->HeadingDir, 'f', 0));
}
```

Figure 3.59 Modified GetOpt Function in postopt.cpp

The function SetOpt is modified by adding the code to set the value of “ElMask2” and “HeadingDir” by the converted values from text to double.

```
void OptDialog::SetOpt(void)
{
    QLineEdit *editu[] = { RovPos1, RovPos2, RovPos3 };
    QLineEdit *editr[] = { RefPos1, RefPos2, RefPos3 };

    mainForm->PosMode = PosMode->currentIndex();
    mainForm->Freq = Freq->currentIndex();
    mainForm->Solution = Solution->currentIndex();
    mainForm->ElMask = ElMask->currentText().toDouble();
    mainForm->ElMask2 = ElMask2->currentText().toDouble();
    mainForm->HeadingDir = HeadingDir->text().toDouble();
}
```

Figure 3.60 Modified SetOpt Function in postopt.cpp

The variable to store the value from “ElMask2” and “HeadingDir” are named “elmin2” and “heading”. They are located under the preopt_t structure and its variable preopt.

```
public:
    preopt_t preopt;
```

Figure 3.61 preopt Variable

```
int MainForm::GetOption(prcopt_t &prcopt, solopt_t &solopt,
                      #ilopt_t &filopt)
{
    char buff[1024], *p;
    int sat, ex;

    // processing options
    prcopt.mode      = PosMode;
    prcopt.soltype   = Solution;
    prcopt.nf        = Freq+1;
    prcopt.navsys    = NavSys;
    prcopt.elim1     = E1Mask+D2R;
    prcopt.elim2     = E1Mask2+D2R;
    prcopt.heading   = HeadingDir;
}
```

Figure 3.62 Modified GetOption Function in postmain.cpp

3.2.4 Experiments

All of the experiments in this section are done using the unmodified version of all RTKLIB application programs.

3.2.4.1 Obtaining the Data

A collection of data is achieved by downloading the logged data from ReachView. After the data is obtained, it must be converted using RTKCONV. Converting the data is necessary because RTKPOST can only process observation and navigation data instead of UBX data.

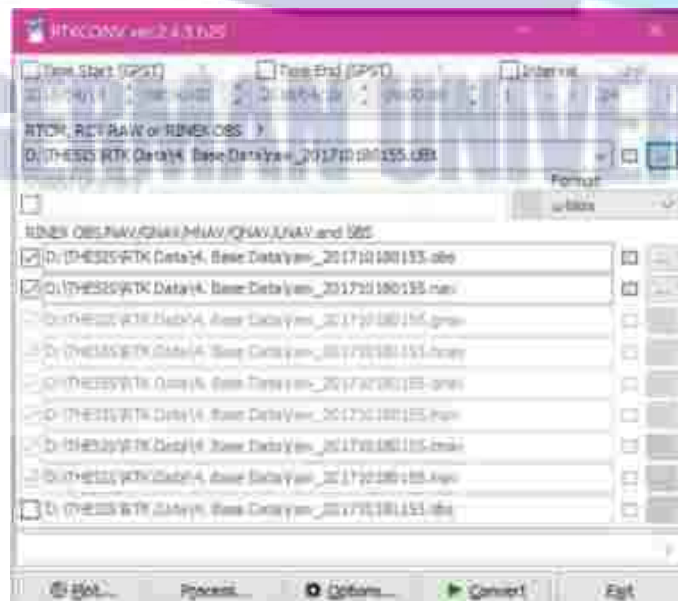


Figure 3.63 Converting UBX Data Using RTKCONV

In options menu, the RTKCONV settings for satellite systems must be adjusted accordingly to the satellite selection that has been previously made in ReachView.

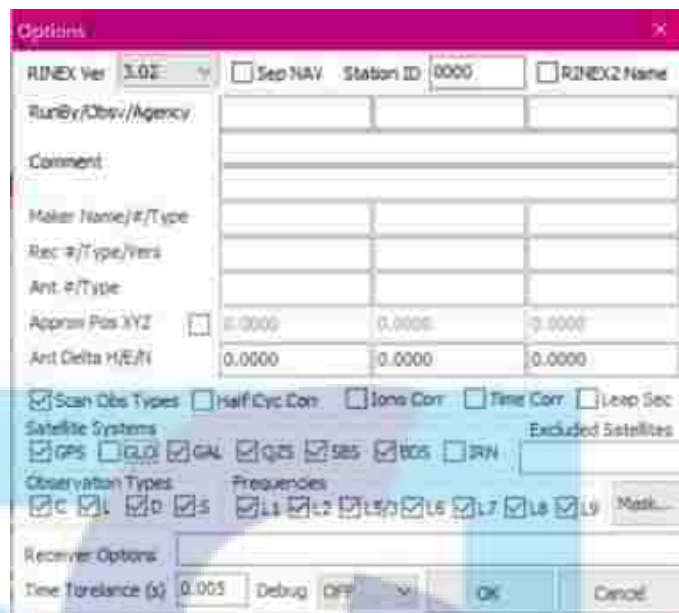


Figure 3.64 RTKCONV Settings

After observation and navigation data has been achieved, the observation files for rover and base station and the navigation file are inputted in RTKPOST.

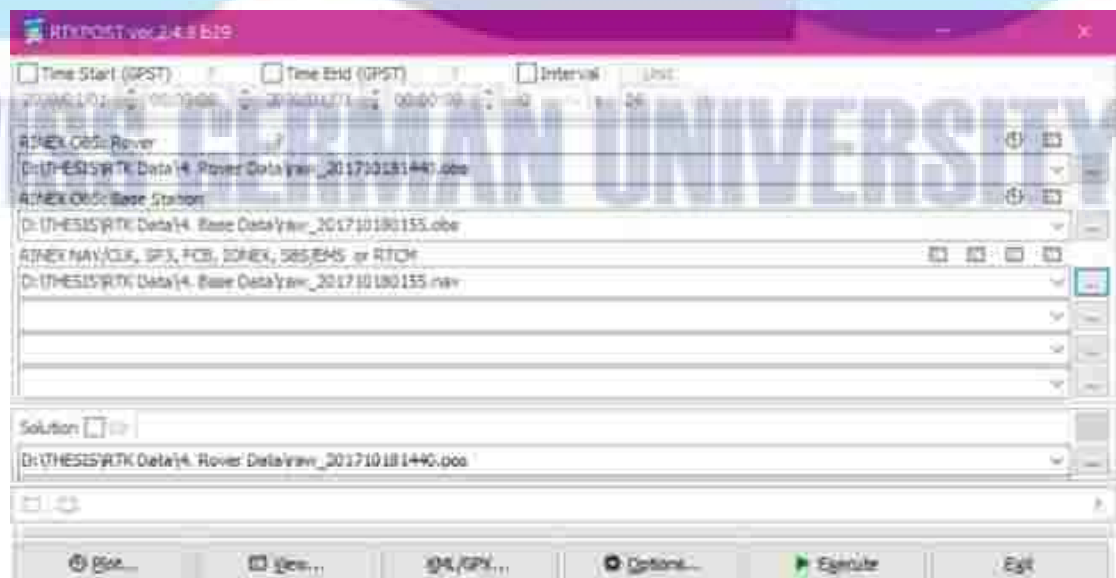


Figure 3.65 Post-processing Data Using RTKPOST

There are several RTKPOST settings that must be adjusted in options menu. The positioning mode is set to kinematic. The elevation mask can be changed ranging from

0° to 70°. The satellite constellations that is desired to be used are ticked. The other settings are left unchanged as default.

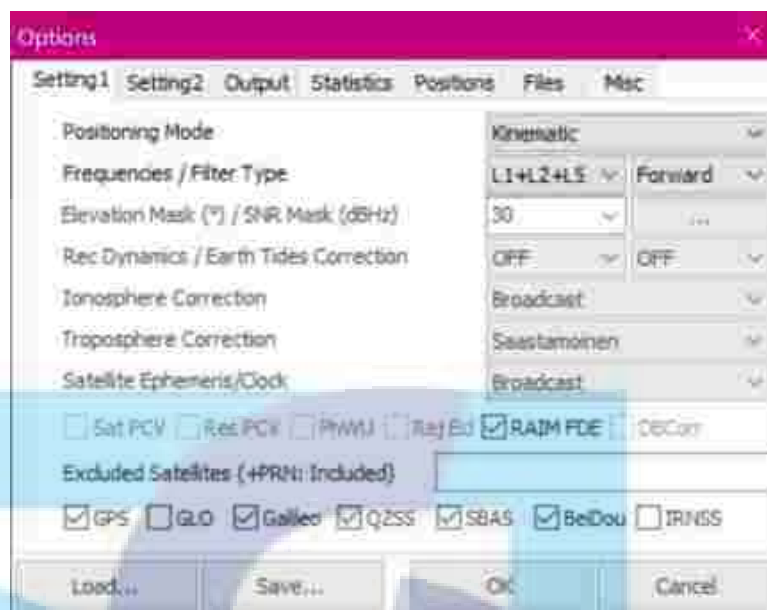


Figure 3.66 RTKPOST Settings (1)

Integer ambiguity resolution is set to fix-and-hold for GPS and Beidou, since GLONASS is unused in the satellite selection in ReachView. The other settings are left unchanged as default.

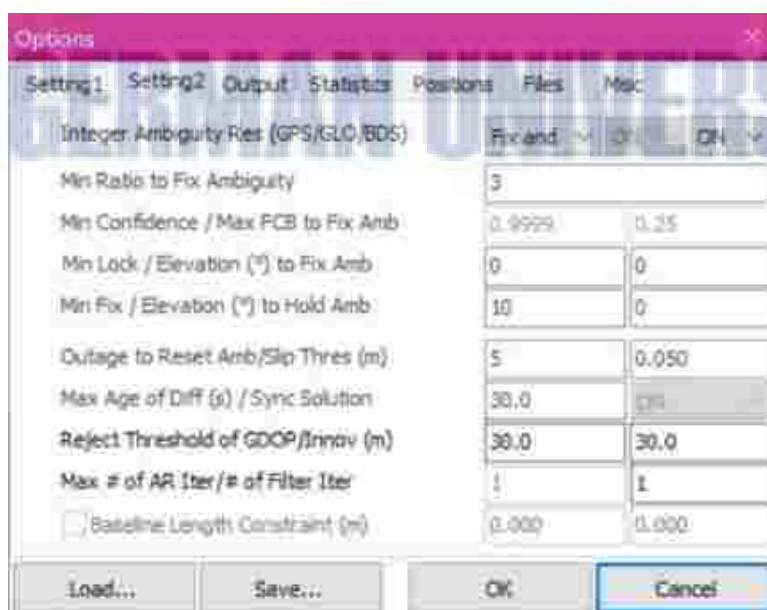


Figure 3.67 RTKPOST Settings (2)

The base station coordinate is inputted based on latitude, longitude, and height obtained from averaging the single position for 30 minutes in ReachView. The other settings are remained unchanged as default.

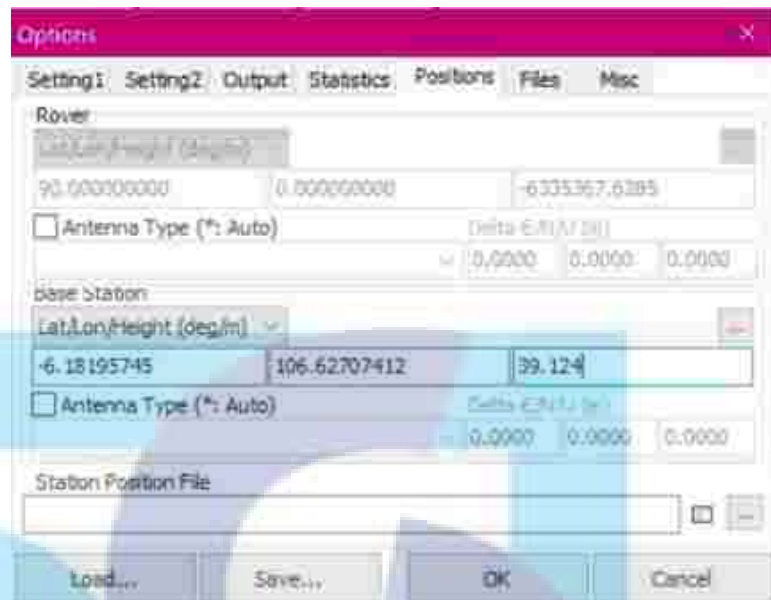


Figure 3.68 RTKPOST Settings (3)

3.2.4.2 Difference Between Single, Fixed, and Float Solutions

The purpose of this experiment is to compare the single, fixed and float solutions. Most of the time, the fixed solutions are more favorable because the number of wavelengths is a whole number, or integer, and the algorithm is constrained to yield a whole number. In the float solutions, the algorithm does not yield acceptable fixed solutions, so the ambiguity is allowed to be a decimal or floating-point number. As for the single solutions, since the solution has not been corrected, the accuracy is expected to be lower than the float solutions.

Using the position data obtained from experiment 3.2.4.3, an average of latitude and longitude of each elevation mask is calculated. After that, the distance for each data to the average point is determined by formula:

$$\text{Distance [n]} = \sqrt{[(\text{latitude}[n] - \overline{\text{latitude}})^2 + (\text{longitude}[n] - \overline{\text{longitude}})^2]}$$

The data dissemination about the distance to average is then plotted and analyzed.

3.2.4.3 Effect of Elevation Mask Angle to RTK Solutions

The purpose of this experiment is to observe the effect of various elevation mask. It is important to know what the optimum elevation mask is, because if the elevation mask is too low, ground multipath can dominates, but if the elevation mask is too high, it is possible that at a certain time, there is not enough satellite to create solution.

For this experiment, GPS + GLONASS + Galileo + QZSS are chosen with update rate of 5 Hz since they are the most popular satellites constellations. GPS and GLONASS are composed from 24 satellites while Galileo is composed from 30 satellites and QZSS is composed from 3 satellites. The rover station receiver is placed next to the base station receiver. The base and rover station receiver are set to log satellite data for 24h. The data for base and rover receiver will be analyzed 6 times by changing only the elevation mask to obtain 6 different solutions in RTKPOST. The solutions are the position data with elevation mask of 0°, 10°, 20°, 30°, 40°, and 50° respectively.

The position data obtained from post-processing can be opened with notepad and copied to Microsoft Excel for the ease of analyzing the data. A graph to represent the effect of changing elevation mask will be made.

3.2.4.4 Relation Between Elevation Angle and SNR

The purpose of this experiment is to investigate whether satellites that are located at low elevation angle deliver more noise than desired signal by investigating the relation between elevation angle and Signal to Noise Ratio (SNR). SNR is usually expressed in terms of decibels. It refers to the ratio of the signal power and noise power in a given bandwidth. In general, SNR increased with the increase of the satellite elevation angle [19]. At low elevation angles, the SNR is low and ground multipath dominates which reduces the signal strength [20]. Using the observation base data obtained from experiment 3.2.4.3, a graph between elevation mask vs SNR is plotted and then the trendline is then determined.

3.2.4.5 Satellite Constellations Selection

The purpose of this experiment is to determine which satellite constellations are the most suitable to be chosen in Tangerang area, because with the right satellite selection, a better positioning result for RTK GNSS can be achieved. This experiment becomes necessary since GLONASS and Beidou by default can't be used together due to frequency issue. There are two satellite constellations configurations that will be tested: GPS + GLONASS + Galileo + SBAS + QZSS and GPS + Beidou + Galileo + SBAS + QZSS. To conduct this experiment, base and rover station receiver are placed side by side and set to log satellite data for 24h for each satellite constellation configuration. By logging the data for 24h, all of the satellites' data that are available in the Tangerang area are recorded. The data is then downloaded and converted using RTKCONV. The solution data is then created using RTKPOST with elevation mask of 30°. After that, the position data is examined in Microsoft Excel for comparing which configuration provides more satellites to be used in Tangerang area.

3.3 Rotary Incremental Encoder

3.3.1 Limitation

Before constructing electrical design and algorithm for the rotary incremental encoder, it is recommended to read the manual book first to know whether the limitation lies in the mechanic or electric side to avoid any problem during the development state later. From the rotary incremental encoder specification, it is known that one rotation generates 1024 pulses. The maximum frequency is 300kHz and the maximum speed is 6000rpm. From the mechanical side, if the rotary incremental encoder is rotated at its maximum speed, it will produce frequency of 102,4kHz.

$$1 \text{ rps} = 60 \text{ rpm}$$

$$1 \text{ rps} \rightarrow 1024 \text{ pulses/second} = 1024 \text{ Hz}$$

$$6000 \text{ rpm} \rightarrow 102,4 \text{ kHz} \rightarrow 9,8\mu\text{s/pulse}$$

From the electric side, the maximum allowable frequency is 300kHz. Therefore, the limitation lies in mechanical side since it has the lower frequency. This mechanical limitation should be taken into consideration when designing the software, so that the software response time should be faster than 9,8μs.

3.3.2 Electrical Design

The electrical design of rotary incremental encoder consists of several components, which are:

Components	Amount
Rotary Incremental Encoder	1
Optocoupler 4N25	3
Resistor 1k Ω	6
Boost Converter XL6009	1
Arduino Mega 2560	1
Power Supply 5V	1
Diode 1N4002	3
USB power bank	1

Table 3.1 Electrical Components List

The boost converter is used to convert 5V supply to 12V, so this circuit only needs one power supply, which is either via Arduino USB connection during normal operating condition or via a power bank when the supply from Arduino USB connection is cut off.

The power bank acts as a back-up power supply, in case the main power supply via Arduino USB connection has a failure. For designing the back-up power supply mechanism, three diodes are used. The first diode is connected between the Arduino USB connector output and 5V Arduino input and the second and third diodes are connected in series between the power bank output and 5V Arduino input. The reason for using 2 diodes in series for the power bank side is so that during normal condition, the voltage supply for the circuit comes only from the Arduino USB connection and the power bank is always in charging position. When the main power supply from Arduino USB connection is suddenly cut off, the power bank will automatically stop charging and it supplies voltage for the circuit, so that the rotary incremental encoder position data won't be lost. The importance of one diode in the Arduino USB connection side is to make sure that when the main voltage is cut off, the power bank will only act as the power supply for the circuit and it doesn't charging anymore.

The optocouplers are used to connect the rotary incremental encoder to Arduino while maintaining two isolated system. The purpose of using optocouplers is to prevent the Arduino from receiving 12V from the rotary incremental encoder output if any error occurs in the rotary incremental encoder circuit. In short, it is used for safety measure.

During normal condition with only Arduino internal pull-up resistor enabled, the time needed for rotary incremental encoder's pulse to reach 99,3% is known to be approximately 500 μ s. Therefore,

$$500\mu\text{s} = 5 \times R \times C$$

$$500\mu\text{s} = 5 \times 20.000\Omega \times C$$

$$C = 5\text{nF}$$

In order to make sure that the rotary incremental encoder pulse will always be detected by the Arduino, the rotary incremental encoder pulse's voltage must always reach minimum $V_{I\text{low}}$ (minimum voltage required by Arduino to be detected as input high), which is 3V. This can be done by making the rise time of rotary incremental encoder pulse faster.

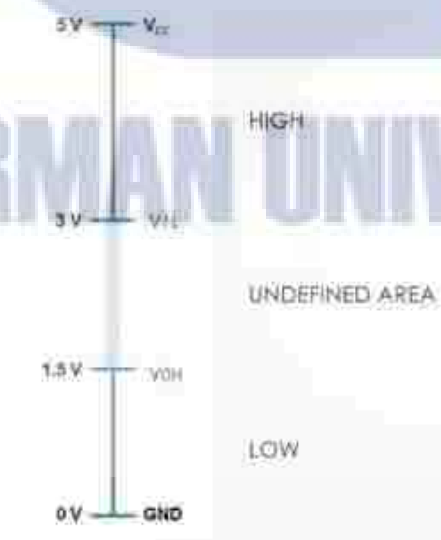


Figure 3.69 Arduino AT Mega 2560 Logic Levels

From the capacitor's charging formula,

$$V_c = V_s \times (1 - e^{-\frac{t}{\tau}})$$

Where

V_c is final condition

V_s is initial condition

t is the elapsed time since the removal of the supply voltage

$\tau = R \times C$ is the time constant of the RC discharging circuit

If the time needed by rotary incremental encoder pulse to reach 5V is $5 \times R \times C$, then the time needed by rotary incremental encoder pulse to reach 3V is:

$$3V = 5V \times (1 - e^{-\frac{t}{\tau}})$$

$$0,4 = e^{-\frac{t}{\tau}}$$

$$\ln(0,4) = -\frac{t}{R \times C}$$

$$t = 0,9 R \times C$$

Suppose the rotary incremental encoder is turned at its maximum speed, therefore, with duty cycle of 50%, the on time of rotary incremental encoder pulse is $4,9\mu s$. To fulfill the V_{low} requirement, it is obligatory that during the on time of rotary incremental encoder pulse, it must have reached 3V to prevent miscount. Therefore, the resistor value must be changed.

$$4,9\mu s = 0,9 R \times 5nF$$

$$R = 1088,88\Omega$$

This is the maximum external pull-up resistor value.

Since the difference between Arduino internal pull-up resistor and external pull-up resistor is significant, the Arduino internal pull-up resistor can be disabled. An external pull-up resistor of $1k\Omega$ is connected in the circuit instead.

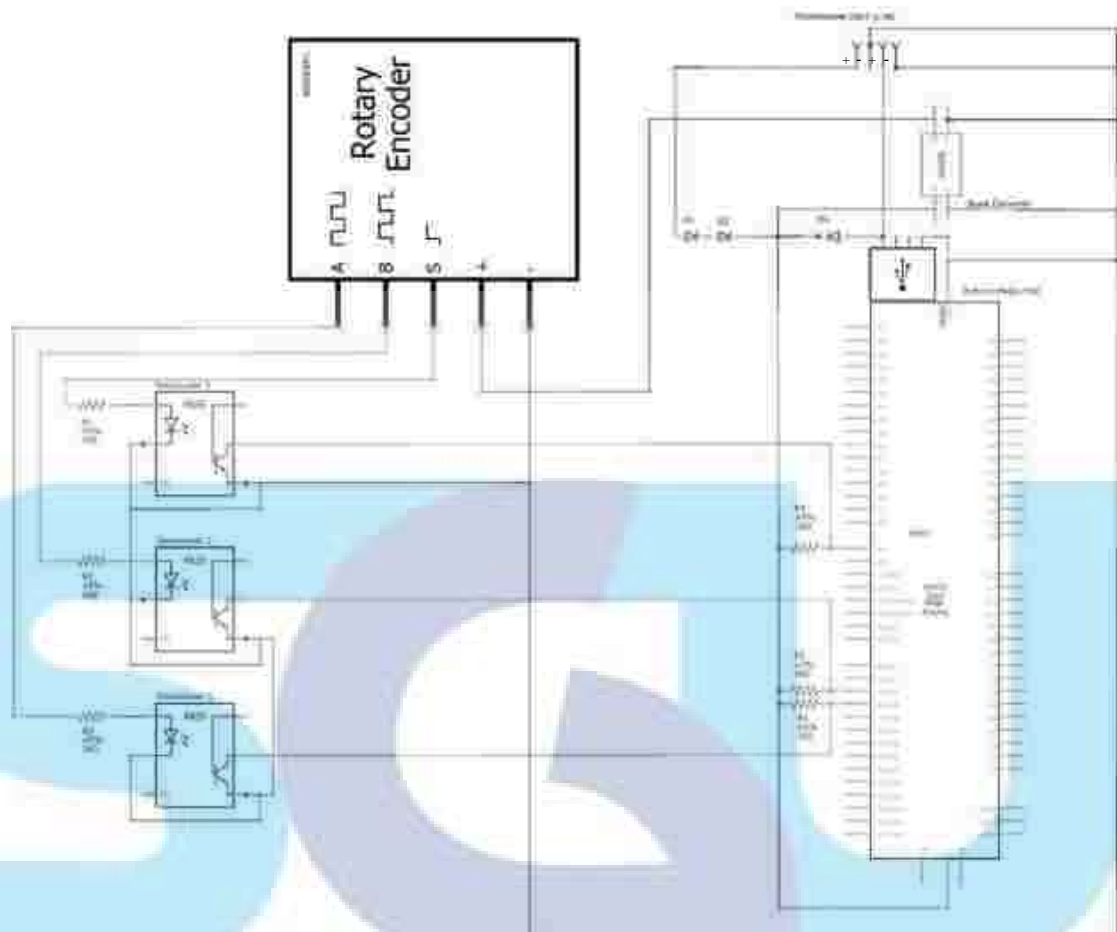


Figure 3.70 Rotary Incremental Encoder Circuit

3.3.3 Algorithm

The rotary incremental encoder is programmed interrupt function in Arduino by incorporating the outputs provided by phase A, B, and Z. The main reason for using interrupt is to make sure no miscount happens. When the Arduino receives an interrupt signal, it will tell the processor to stop what it is doing at the moment and switch to handle the interrupt function task. In this case, the interrupt function tasks are doEncoderA, doEncoderB, and doEncoderZ. Whenever there is a change of signal state in either pin 2, 3, or 21 the interrupt function task will be run accordingly.

```
// encoder channel A interrupt
attachInterrupt(digitalPinToInterrupt(2), doEncoderA, FALLING);

// encoder channel B interrupt
attachInterrupt(digitalPinToInterrupt(3), doEncoderB, FALLING);

// encoder channel Z interrupt
attachInterrupt(digitalPinToInterrupt(21), doEncoderZ, FALLING);
```

Figure 3.71 Interrupt Function

In addition to interrupt function, Arduino sleep function is used as well. The benefit of using sleep is to make the energy use more efficient. After the Arduino has finished doing the task in the main loop and there is no interrupt present, the Arduino is set to idle to decrease its energy consumption.

In the main loop, the Arduino's task is mainly to update the red led status. If the homing procedure is not yet done or if a failure occurs, then the red led will be turned on. If the homing procedure is already done and there is no failure, then the red led will be turned off. The failure can happen either because of the inconsistency of the rotary incremental encoder count value when Z signal is present or when Z signal does not appear when it should appear. This failure detection is a safety measure for the autonomous car. The failure detection is implemented to prevent the autonomous car from turning the electric power steering in the wrong direction due to miscounting from the rotary incremental encoder. Before the electric power steering is ready to be used, there is a homing procedure to make sure that initially, the mechanical position is already in home position. When the electric power steering is already turned on, the validation method is available as well.

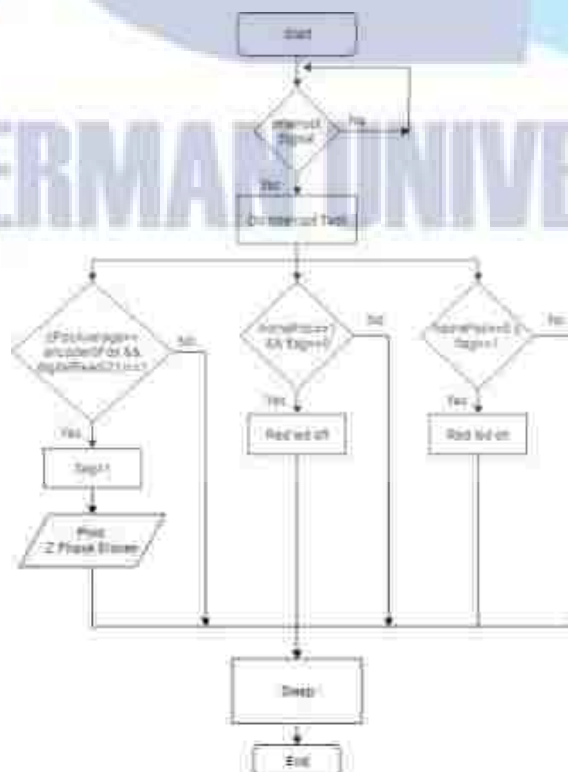


Figure 3.72 Main Loop Algorithm

During programming, bitRead is used instead of the standard digitalRead to make the Arduino processing time faster. The main benefit of using bitRead is that it manipulates the port directly, thus increasing the processing speed of Arduino.

When there is a signal state change from low to high on phase A, the state of phase B will be checked to determine which way the rotary incremental encoder is turning. If the state of phase B is low, the rotary incremental encoder is turning clockwise, otherwise counter-clockwise.

When there is a signal state change from high to low on phase A, the state of phase B is checked. If the state of phase B is high, the rotary incremental encoder is turning clockwise, otherwise counter-clockwise.

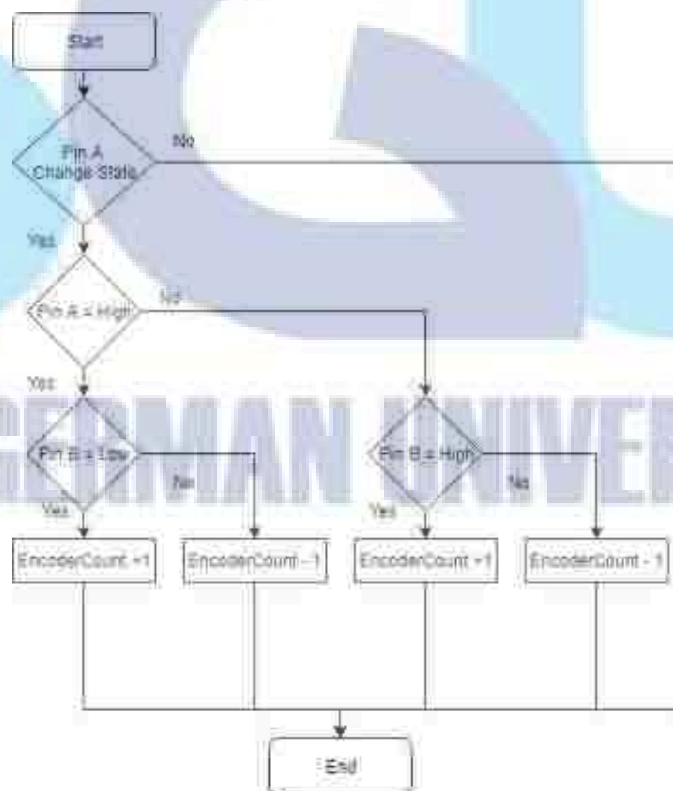


Figure 3.73 Phase A Interrupt Algorithm

When there is a signal state change from low to high on phase B, the state of phase A will be checked to determine which way the rotary incremental encoder is turning. If the state of phase A is high, the rotary incremental encoder is turning clockwise, otherwise counter-clockwise.

When there is a signal state change from high to low on phase B, the state of phase A is checked. If the state of phase A is low, the rotary incremental encoder is turning clockwise, otherwise counter-clockwise.

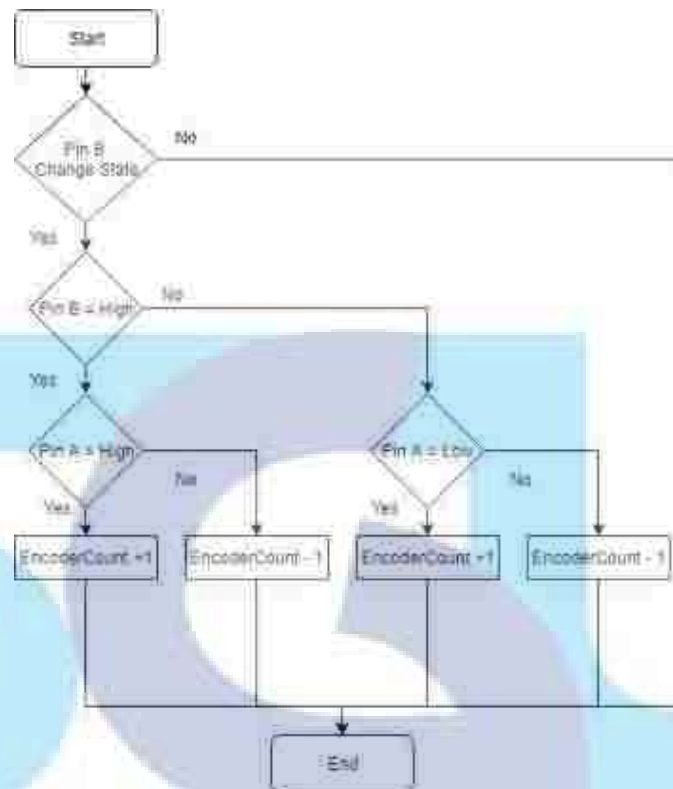


Figure 3.74 Phase B Interrupt Algorithm

The other function task, which is doEncoderZ is used to validate the rotary incremental encoder count value. If the validation fails, the user will be notified. The phase Z will only produce one pulse for each rotation and it will always be in the same exact location. In this case, the phase Z will be used for homing procedure and miscount notification during running.

A homing procedure is the mandatory procedure to do before running the main program to make sure that the mechanical position is in the home position in the first place. When the rotary incremental encoder detects Z signal, the value of current rotary incremental encoder position will be stored in a home register. During homing, 3 rotary incremental encoder position values when Z signal is present will be compared. If they are the same, then the homing procedure is complete and the main program is allowed to be run. Then, during running, when the Z signal is present, the rotary incremental encoder current

position will be compared to the value in the home register to make sure there is no miscount.

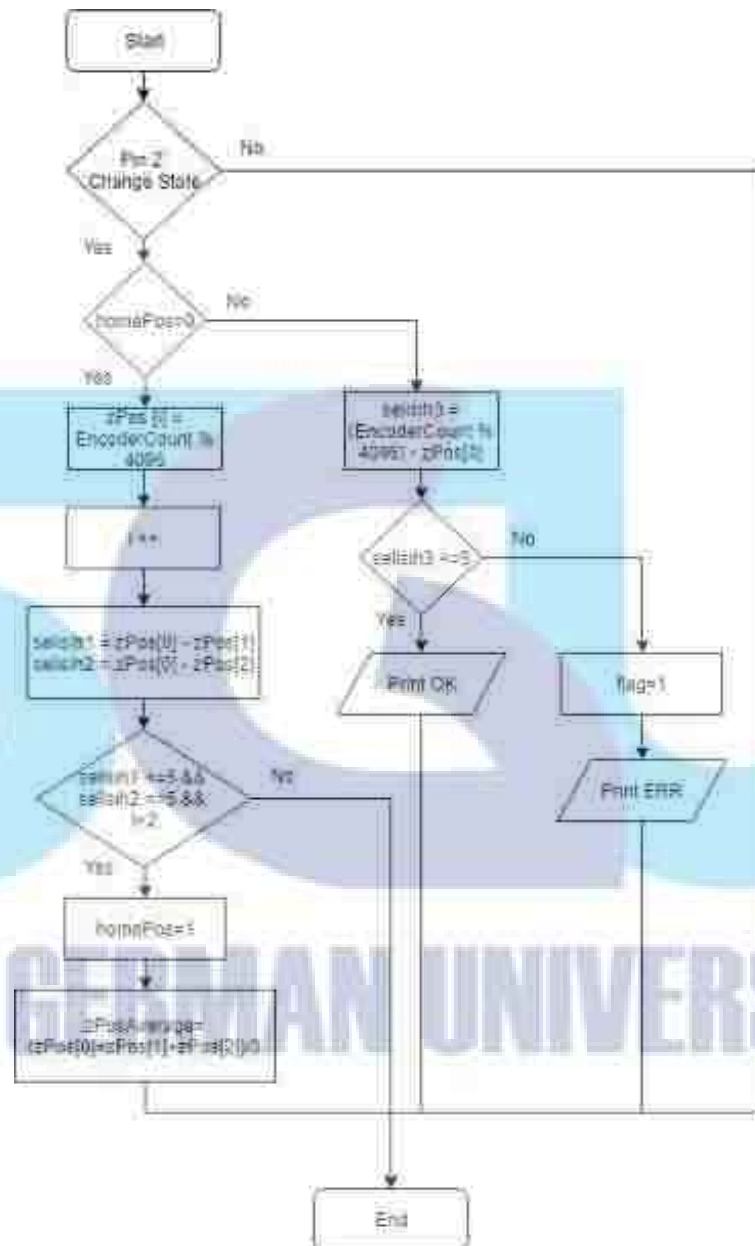


Figure 3.75 Phase Z Interrupt Algorithm