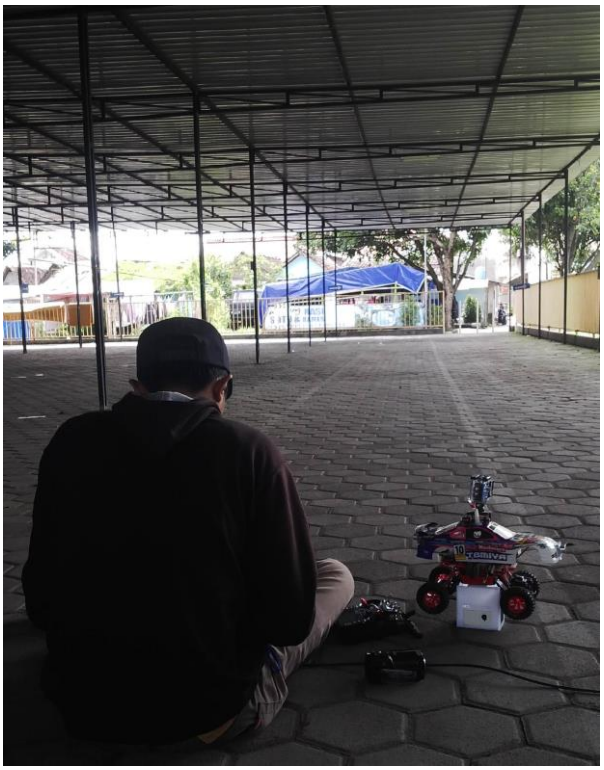
LAMPIRAN

Lampiran 1 Tempat Uji Coba Alat

Gambar 40. Uji Coba Alat di Lapangan dekat SMA Taman Siswa Madya

Tamansiswa



Gambar 41. Uji Coba Alat di Tempat Parkir LPM Pendapa

Lampiran 2 Gambar Alat

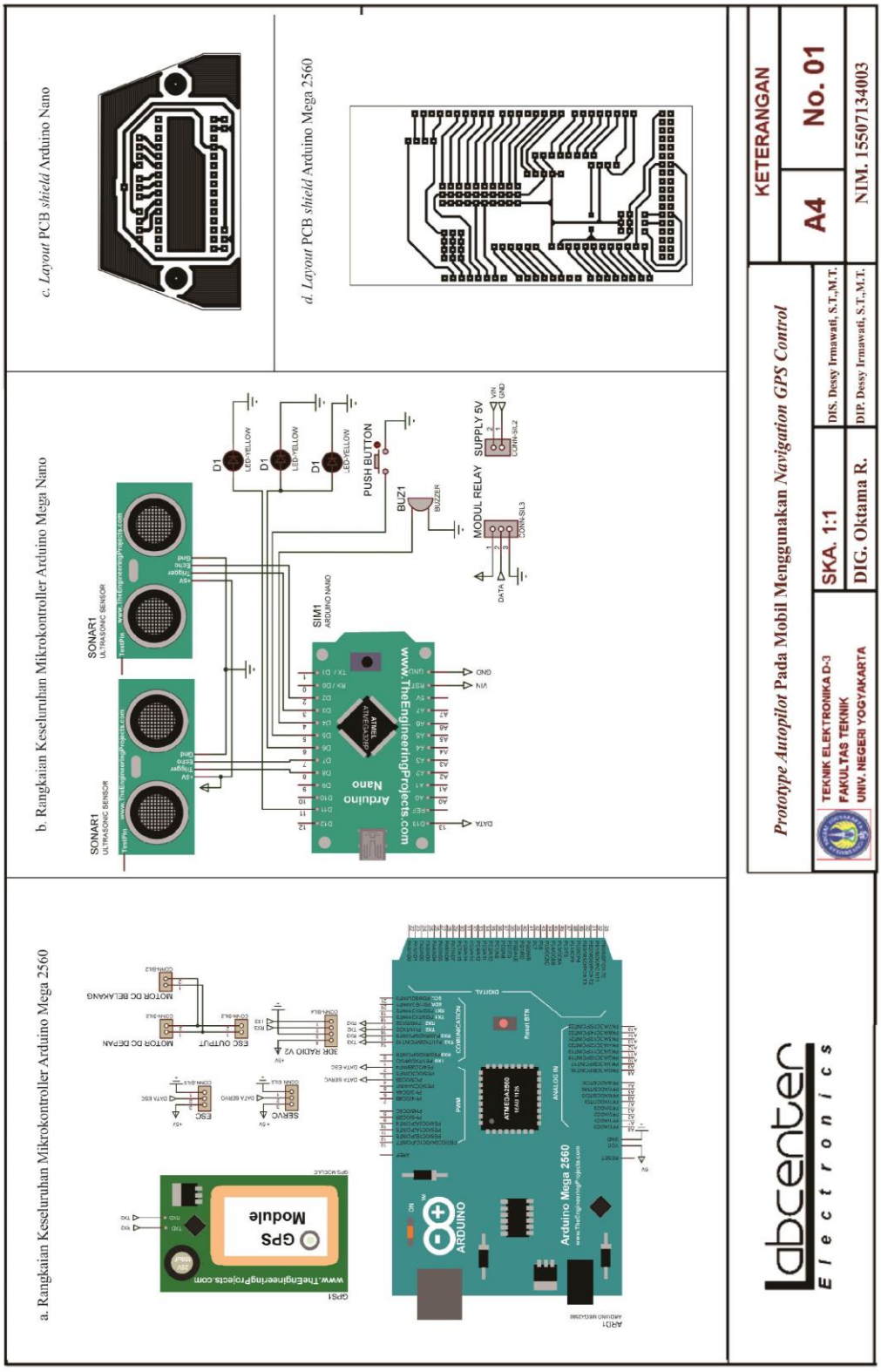
Gambar 42. Tampilan keseluruhan alat dari samping

Gambar 43. Tampilan keseluruhan alat dari depan

Lampiran 3 Pengujian unjuk kerja bagian misi

Gambar 44. Pengujian unjuk kerja bagian misi

Lampiran 4 Skematik Rangkaian



Gambar 45. Skematik Rangkaian

Lampiran 5 Program Arduino Nano

```
#define SECOND 1000UL
#define MINUTE (SECOND * 60UL)

int trigPin_1 = 3; //Trig - green Jumper
int echoPin_1 = 2; //Echo - yellow
Jumper
long duration_1, cm_1;

int trigPin_2 = 7; //Trig - green Jumper
int echoPin_2 = 8; //Echo - yellow
Jumper
long duration_2, cm_2;

const int Relay = 13;
const int LED = 6;
const int LED2 = 11;
const int Button = 5;
const int Buzzer = 4;
int buttonState;

void setup() {
//Serial Port begin
Serial.begin (9600);
//Define inputs and outputs
pinMode(trigPin_1, OUTPUT);
pinMode(echoPin_1, INPUT);

pinMode(trigPin_2, OUTPUT);
pinMode(echoPin_2, INPUT);

pinMode(Relay, OUTPUT);
pinMode(LED, OUTPUT);
pinMode(LED2, OUTPUT);
pinMode(Buzzer, OUTPUT);
pinMode(Button, INPUT_PULLUP);

digitalWrite(LED2, HIGH);
}
void loop()
{
  Alarm();
  sensor_1();
  sensor_2();
  Jarak();
}
```

```
void Jarak ()
{
   if(cm_1 >= 20&&cm_2 >= 20)
   {
    Relay_ON();
   }
   else if(cm_1 < 20&&cm_2 < 20)
   {
    Relay_OFF();
   }
   else
   {
    Relay_OFF();
   }
}
void Relay_ON()
{
  digitalWrite(Relay, HIGH);
  digitalWrite(LED, LOW);
  diam();
}

void Relay_OFF()
{
  digitalWrite(Relay, LOW);
  digitalWrite(LED, HIGH);
  nada_();
}
void sensor_1()
{
// Sensor dipicu oleh pulsa HIGH dari 10us
atau lebih.
// Berikan pulsa LOW pendek terlebih dahulu
untuk memastikan pulsa HIGH bersih:
digitalWrite(trigPin_1, LOW);
delayMicroseconds(2);
digitalWrite(trigPin_1, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin_1, LOW);
// Baca sinyal dari sensor: pulsa HIGH
// duration adalah waktu (dalam mikrodetik)
dari pengirim
// Dari ping untuk penerimaan Echo off dari
sebuah objek.
pinMode(echoPin_1, INPUT);
duration_1 = pulseIn(echoPin_1, HIGH);
// convert jarak kedalam cm
cm_1 = duration_1 / 58.2;
Serial.print("Jarak 1 =");
Serial.print(cm_1);
Serial.print("cm");
Serial.println();
//delay(250);
}
```

```
void Alarm()
{
  buttonState = digitalRead(Button);
  if(buttonState == LOW)
    {
    digitalWrite(Relay, HIGH);
    digitalWrite(LED, LOW);
    nada2_();
    delay(10*SECOND);
    }
  else
  { digitalWrite(Relay, HIGH);diam();}
}

void nada_DO()
{
   tone(Buzzer, 262);
}
void nada_RE()
{
   tone(Buzzer, 294);
}
void nada_MI()
{
   tone(Buzzer, 330);
}
void nada_FA()
{
   tone(Buzzer, 349);
}
void nada_SOL()
{
   tone(Buzzer, 395);
}
void nada_LA()
{
   tone(Buzzer, 440);
}
void diam()
{
   noTone(Buzzer);
}
void nada_()
{
nada_DO ();delay(250);nada_RE();
delay(250);
nada_MI ();delay(250);nada_FA();
delay(250);
nada_MI ();delay(500);nada_FA();
delay(250);
diam();delay(250);}
void nada2_()
{
nada_SOL ();delay(250);nada_LA();
delay(250);
nada_SOL ();delay(250);nada_LA();
delay(250);
nada_SOL ();delay(250);nada_LA();
delay(250);
diam();delay(250);}
```

```
void sensor_2()
{
// Sensor dipicu oleh pulsa HIGH dari 10us
atau lebih.
// Berikan pulsa LOW pendek terlebih dahulu
untuk memastikan pulsa HIGH bersih:
digitalWrite(trigPin_2, LOW);
delayMicroseconds(2);
digitalWrite(trigPin_2, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin_2, LOW);
// Baca sinyal dari sensor: pulsa HIGH
// duration adalah waktu (dalam mikrodetik)
dari pengirim
// Dari ping untuk penerimaan Echo off dari
sebuah objek.
pinMode(echoPin_2, INPUT);
duration_2 = pulseIn(echoPin_2, HIGH);
// convert jarak kedalam cm
cm_2 = duration_2 / 58.2;
Serial.print("Jarak 2 =");
Serial.print(cm_2);
Serial.print("cm");
Serial.println();
//delay(250);
}
```

Lampiran 6 Program Arduino Mega

```
#define THISFIRMWARE "ArduRover v2.43"
#include <math.h>
#include <stdarg.h>
#include <stdio.h>
// Libraries
#include <AP_Common.h>
#include <AP_Progmem.h>
#include <AP_HAL.h>
#include <AP_Menu.h>
#include <AP_Param.h>
#include <AP_GPS.h>         // ArduPilot GPS library
#include <AP_ADC.h>         // ArduPilot Mega Analog to Digital Converter Library
#include <AP_ADC_AnalogSource.h>
#include <AP_Baro.h>
#include <AP_Compass.h>     // ArduPilot Mega Magnetometer Library
#include <AP_Math.h>        // ArduPilot Mega Vector/Matrix math Library
#include <AP_InertialSensor.h> // Inertial Sensor (uncalibated IMU) Library
#include <AP_AHRS.h>        // ArduPilot Mega DCM Library
#include <PID.h>            // PID library
#include <RC_Channel.h>     // RC Channel Library
#include <AP_RangeFinder.h> // Range finder library
#include <Filter.h>
        // Filter library
#include <Butter.h>
        // Filter library - butterworth filter
#include <AP_Buffer.h>      // FIFO buffer library
#include <ModeFilter.h>
        // Mode Filter from Filter library
#include <AverageFilter.h>      // Mode Filter from Filter library
#include <AP_Relay.h>       // APM relay
#include <AP_Mount.h>
        // Camera/Antenna mount
#include <AP_Camera.h>
        // Camera triggering
#include <GCS_MAVLink.h>    // MAVLink GCS definitions
#include <AP_Airspeed.h>    // needed for AHRS build
#include <AP_Vehicle.h>     // needed for AHRS build
#include <memcheck.h>
```

```
    #include <DataFlash.h>
    #include <AP_RCMapper.h>        // RC input mapping library
    #include <SITL.h>
    #include <AP_Scheduler.h>       // main loop scheduler
    #include <stdarg.h>
    #include <AP_Navigation.h>
    #include <APM_Control.h>
    #include <AP_L1_Control.h>

    #include <AP_HAL_AVR.h>
    #include <AP_HAL_AVR_SITL.h>
    #include <AP_HAL_PX4.h>
    #include <AP_HAL_MPNG.h>
    #include <AP_HAL_FLYMAPLE.h>
    #include <AP_HAL_Linux.h>
    #include <AP_HAL_Empty.h>
    #include "compat.h"

    #include <AP_Notify.h>      // Notify library
    #include <AP_BattMonitor.h> // Battery monitor library

    // Configuration
    #include "config.h"
```

```
    // Local modules
    #include "defines.h"
    #include "Parameters.h"
    #include "GCS.h"

    #include <AP_Declination.h> // ArduPilot Mega Declination Helper Library

    AP_HAL::BetterStream* cliSerial;

    const AP_HAL::HAL& hal = AP_HAL_BOARD_DRIVER;

    // this sets up the parameter table, and sets the default values. This
    // must be the first AP_Param variable declared to ensure its
    // constructor runs before the constructors of the other AP_Param
    // variables
    AP_Param param_loader(var_info, WP_START_BYTE);
    static const AP_InertialSensor::Sample_rate ins_sample_rate = AP_InertialSensor::RATE_50HZ;
    static Parameters      g;
    static AP_Scheduler scheduler;
```

```
    // mapping between input channels
    static RCMapper rcmap;
    // primary control channels
    static RC_Channel *channel_steer;
    static RC_Channel *channel_throttle;
    static RC_Channel *channel_learn;
    static void update_events(void);
    void gcs_send_text_fmt(const
prog_char_t *fmt, ...);
    static void
print_mode(AP_HAL::BetterStream *port,
uint8_t mode);
    #if CONFIG_HAL_BOARD ==
HAL_BOARD_APM1
    static DataFlash_APM1 DataFlash;
    #elif CONFIG_HAL_BOARD ==
HAL_BOARD_APM2
    static DataFlash_APM2 DataFlash;
    #elif CONFIG_HAL_BOARD ==
HAL_BOARD_MPNG
    static DataFlash_MPNG DataFlash;
    #elif CONFIG_HAL_BOARD ==
HAL_BOARD_AVR_SITL
    //static DataFlash_File
DataFlash("/tmp/APMlogs");
    static DataFlash_SITL DataFlash;
    #elif CONFIG_HAL_BOARD ==
HAL_BOARD_PX4
    static DataFlash_File
DataFlash("/fs/microsd/APM/logs");
    #elif CONFIG_HAL_BOARD ==
HAL_BOARD_LINUX
    static DataFlash_File DataFlash("logs");
    #else
    DataFlash_Empty DataFlash;
    #endif
    access should be through this pointer.
    static GPS        *g_gps;
    static AP_Int8                *modes
= &g.mode1;
    #if CONFIG_HAL_BOARD ==
HAL_BOARD_APM1
    static AP_ADC_ADS7844 adc;
    #endif
    #if CONFIG_COMPASS ==
AP_COMPASS_PX4
    static AP_Compass_PX4 compass;
    #elif CONFIG_COMPASS ==
AP_COMPASS_HMC5843
    static AP_Compass_HMC5843
compass;
    #elif CONFIG_COMPASS ==
AP_COMPASS_HIL
    static AP_Compass_HIL compass;
    #else
```

```
      #error Unrecognized
CONFIG_COMPASS setting
      #endif

      // GPS selection
      #if   GPS_PROTOCOL ==
GPS_PROTOCOL_AUTO
      AP_GPS_Auto
g_gps_driver(&g_gps);

      #elif GPS_PROTOCOL ==
GPS_PROTOCOL_NMEA
      AP_GPS_NMEA     g_gps_driver;

      #elif GPS_PROTOCOL ==
GPS_PROTOCOL_SIRF
      AP_GPS_SIRF     g_gps_driver;

      #elif GPS_PROTOCOL ==
GPS_PROTOCOL_UBLOX
      AP_GPS_UBLOX    g_gps_driver;

      #elif GPS_PROTOCOL ==
GPS_PROTOCOL_MTK
      AP_GPS_MTK      g_gps_driver;

      #elif GPS_PROTOCOL ==
GPS_PROTOCOL_MTK19
      AP_GPS_MTK19    g_gps_driver;

      #elif GPS_PROTOCOL ==
GPS_PROTOCOL_NONE
      AP_GPS_None     g_gps_driver;

      #elif GPS_PROTOCOL ==
GPS_PROTOCOL_HIL
      AP_GPS_HIL      g_gps_driver;

      #else
        #error Unrecognised
GPS_PROTOCOL setting.
      #endif // GPS PROTOCOL

      #if CONFIG_INS_TYPE ==
CONFIG_INS_MPU6000
      AP_InertialSensor_MPU6000 ins;
      #elif CONFIG_INS_TYPE ==
CONFIG_INS_MPU6000_I2C
      AP_InertialSensor_MPU6000_I2C ins;
      #elif CONFIG_INS_TYPE ==
CONFIG_INS_PX4
      AP_InertialSensor_PX4 ins;
      #elif CONFIG_INS_TYPE ==
CONFIG_INS_HIL
      AP_InertialSensor_HIL ins;
```

```
    #elif CONFIG_INS_TYPE ==
CONFIG_INS_FLYMAPLE
    AP_InertialSensor_Flymaple ins;
    #elif CONFIG_INS_TYPE ==
CONFIG_INS_L3G4200D
    AP_InertialSensor_L3G4200D ins;
    #elif CONFIG_INS_TYPE ==
CONFIG_INS_OILPAN
    AP_InertialSensor_Oilpan ins( &adc );
    #else
      #error Unrecognised
CONFIG_INS_TYPE setting.
    #endif // CONFIG_INS_TYPE
    AP_AHRS_DCM ahrs(ins, g_gps);
    static AP_L1_Control
L1_controller(ahrs);
    // selected navigation controller
    static AP_Navigation *nav_controller =
&L1_controller;
    static AP_SteerController
steerController(ahrs);

    #if CONFIG_HAL_BOARD ==
HAL_BOARD_AVR_SITL
    SITL sitl;
    #endif
    GCS_MAVLINK       gcs0;
    GCS_MAVLINK       gcs3;
```

```
    AP_HAL::AnalogSource
*rssi_analog_source;
    AP_HAL::AnalogSource *vcc_pin;
    static AP_RangeFinder_analog sonar;
    static AP_RangeFinder_analog sonar2;
    // relay support
    AP_Relay relay;
    // Camera
    #if CAMERA == ENABLED
    static AP_Camera camera(&relay);
    #endif
    static struct    Location current_loc;
    #if MOUNT == ENABLED
    camera_mount(&current_loc, g_gps,
ahrs, 0);
    #endif
    static bool usb_connected;
    enum mode   control_mode       =
INITIALISING;
    uint8_t          oldSwitchPosition;
    static int16_t rc_override[8] =
{0,0,0,0,0,0,0,0};
    // A flag if GCS joystick control is in
use
    static bool rc_override_active = false;
    static uint8_t  ground_start_count
          = 5;
```

```
    // on the ground or in the air.  Used to
decide if a ground start is appropriate if we
    // booted with an air start.
    static int16_t     ground_start_avg;
    static int32_t          gps_base_alt;
    const             float radius_of_earth
          = 6378100;          // meters
from AHRS
    static bool have_position;
    static bool rtl_complete = false;
    static uint8_t  nav_command_index;
    static uint8_t
          non_nav_command_index;
    static uint8_t  nav_command_ID
                    = NO_COMMAND;
    static uint8_t  non_nav_command_ID
          = NO_COMMAND;
    // ground speed error in m/s
    static float      groundspeed_error;
    // 0-(throttle_max - throttle_cruise) :
throttle nudge in Auto mode using top 1/2 of
throttle stick travel
    static int16_t     throttle_nudge = 0;
    static uint8_t receiver_rssi;
    static uint32_t last_heartbeat_ms;
    static struct {
       uint8_t detected_count;
       float turn_angle;
```

```
    uint16_t sonar1_distance_cm;
    uint16_t sonar2_distance_cm;
obstacle, in milliseconds
    uint32_t detected_time_ms;
} obstacle;
triggered to start
static bool auto_triggered;
static AP_BattMonitor battery;
static float lateral_acceleration;
static float wp_distance;
static int32_t wp_totalDistance;
static uint8_t              event_id;
static int32_t              event_timer;
static uint16_t             event_delay;
static int16_t              event_repeat = 0;
static int16_t              event_value;
static int16_t
      event_undo_value;
static int32_t  condition_value;
static int32_t  condition_start;
static int16_t              condition_rate;
AP_Common
static struct    Location home;
static bool      home_is_set;
static struct    Location prev_WP;
static struct    Location next_WP;
static struct    Location guided_WP;
next_nav_command;
```

```
    static struct    Location
next_nonnav_command;
    static float G_Dt         = 0.02;
    static int32_t perf_mon_timer;
    // The maximum main loop execution
time recorded in the current performance
monitoring interval
    static uint32_t             G_Dt_max;
    // The number of gps fixes recorded in
the current performance monitoring interval
    static uint8_t gps_fix_count = 0;
    static uint32_t
          fast_loopTimer_us;
    // Number of milliseconds used in last
main loop cycle
    static uint32_t
          delta_us_fast_loop;
    // Counter of main loop executions.
Used for performance monitoring and
failsafe processing
    static uint16_t
          mainLoop_count;
    // set if we are driving backwards
    static bool in_reverse;
    ////////////////////////////////////////////////////////
//////////////////////
     scheduler table - all regular tasks
should be listed here, along
```

```
     with how often they should be called
(in 20ms units) and the maximum
     time they are expected to take (in
microseconds)
     */
    static const AP_Scheduler::Task
scheduler_tasks[] PROGMEM = {
          { read_radio,           1,   1000 },
        { ahrs_update,          1,   6400 },
        { read_sonars,          1,   2000 },
        { update_current_mode,  1,   1000 },
        { set_servos,           1,   1000 },
        { update_GPS,           5,   2500 },
        { navigate,             5,   1600 },
        { update_compass,       5,   2000 },
        { update_commands,      5,   1000 },
        { update_logging,       5,   1000 },
        { gcs_retry_deferred,   1,   1000 },
        { gcs_update,           1,   1700 },
        { gcs_data_stream_send, 1,   3000 },
        { read_control_switch,  15,  1000 },
        { read_trim_switch,     5,   1000 },
        { read_battery,         5,   1000 },
        { read_receiver_rssi,   5,   1000 },
        { update_events,        15,  1000 },
        { check_usb_mux,        15,  1000 },
        { mount_update,         1,    600 },
        { gcs_failsafe_check,   5,    600 },
```

```
        { compass_accumulate,    1,    900 },
        { update_notify,         1,    300 },
        { one_second_loop,       50,   3000 }
    };
    void setup() {
        memcheck_init();
        cliSerial = hal.console;
        // load the default values of variables listed in var_info[]
        AP_Param::setup_sketch_defaults();
        // rover does not use arming nor pre-arm checks
        AP_Notify::flags.armed = true;
        AP_Notify::flags.pre_arm_check = true;
        AP_Notify::flags.failsafe_battery = false;
        notify.init();
        battery.init();

        rssi_analog_source = hal.analogin->channel(ANALOG_INPUT_NONE);
        vcc_pin = hal.analogin->channel(ANALOG_INPUT_BOARD_VCC);

        init_ardupilot();
```

```
        // initialise the main loop scheduler
        scheduler.init(&scheduler_tasks[0], sizeof(scheduler_tasks)/sizeof(scheduler_tasks[0]));
    }
    /*
     loop() is called rapidly while the sketch is running
     */
    void loop()
    {
        // wait for an INS sample
        if (!ins.wait_for_sample(1000)) {
            return;
        }
        uint32_t timer = hal.scheduler->micros();
        delta_us_fast_loop  = timer - fast_loopTimer_us;
        G_Dt                = delta_us_fast_loop * 1.0e-6f;
        fast_loopTimer_us   = timer;

          if (delta_us_fast_loop > G_Dt_max)
                    G_Dt_max = delta_us_fast_loop;
```

```
        mainLoop_count++;

        // tell the scheduler one tick has
passed
        scheduler.tick();

        scheduler.run(19500U);
    }
    // update AHRS system
    static void ahrs_update()
    {
    #if HIL_MODE !=
HIL_MODE_DISABLED
        // update hil before AHRS update
        gcs_update();
    #endif
        // when in reverse we need to tell
AHRS not to assume we are a
        // 'fly forward' vehicle, otherwise it
will see a large
        // discrepancy between the mag and
the GPS heading and will try to
        // correct for it, leading to a large yaw
error
        ahrs.set_fly_forward(!in_reverse);

        ahrs.update();
```

```
        if (g.log_bitmask &
MASK_LOG_ATTITUDE_FAST)
            Log_Write_Attitude();

        if (g.log_bitmask &
MASK_LOG_IMU)
            DataFlash.Log_Write_IMU(ins);
    }
    /*
      update camera mount - 50Hz
     */
    static void mount_update(void)
    {
    #if MOUNT == ENABLED
        camera_mount.update_mount_posit
ion();
    #endif
    #if CAMERA == ENABLED
      camera.trigger_pic_cleanup();
    #endif
    }
    /*
      check for GCS failsafe - 10Hz
     */
    static void gcs_failsafe_check(void)
    {
        if (g.fs_gcs_enabled) {
```

```
failsafe_trigger(FAILSAFE_EVENT_GCS,
last_heartbeat_ms != 0 && (millis() -
last_heartbeat_ms) > 2000);
        }
    }
    /*
     if the compass is enabled then try to
accumulate a reading
     */
    static void compass_accumulate(void)
    {
        if (g.compass_enabled) {
            compass.accumulate();
        }
    }
    /*
     check for new compass data - 10Hz
     */
    static void update_compass(void)
    {
        if (g.compass_enabled &&
compass.read()) {
            ahrs.set_compass(&compass);
            // update offsets
            compass.null_offsets();
            if (g.log_bitmask &
MASK_LOG_COMPASS) {
                Log_Write_Compass();
            }
        } else {
            ahrs.set_compass(NULL);
        }
    }
    /*
     log some key data - 10Hz
     */
    static void update_logging(void)
    {
        if ((g.log_bitmask &
MASK_LOG_ATTITUDE_MED) &&
!(g.log_bitmask &
MASK_LOG_ATTITUDE_FAST))
            Log_Write_Attitude();
        if (g.log_bitmask &
MASK_LOG_CTUN)
            Log_Write_Control_Tuning();
        if (g.log_bitmask &
MASK_LOG_NTUN)
            Log_Write_Nav_Tuning();
    }
    /*
     update aux servo mappings
     */
    static void update_aux(void)
    {
```

```
    #if CONFIG_HAL_BOARD ==
HAL_BOARD_PX4
        update_aux_servo_function(&g.rc_5,
&g.rc_6, &g.rc_7, &g.rc_8, &g.rc_9,
&g.rc_10, &g.rc_11, &g.rc_12);
    #elif CONFIG_HAL_BOARD ==
HAL_BOARD_APM2
        update_aux_servo_function(&g.rc_5,
&g.rc_6, &g.rc_7, &g.rc_8, &g.rc_10,
&g.rc_11);
    #else
        update_aux_servo_function(&g.rc_5,
&g.rc_6, &g.rc_7, &g.rc_8);
    #endif
        enable_aux_servos();

    #if MOUNT == ENABLED
        camera_mount.update_mount_type();
    #endif
    }
    /*
     once a second events
     */
    static void one_second_loop(void)
    {
          if (g.log_bitmask &
MASK_LOG_CURRENT)
                    Log_Write_Current();
          // send a heartbeat
          gcs_send_message(MSG_HEART
BEAT);
        // allow orientation change at runtime
to aid config
        ahrs.set_orientation();
        set_control_channels();
        // cope with changes to aux functions
        update_aux();
    #if MOUNT == ENABLED
        camera_mount.update_mount_type();
    #endif
        // cope with changes to mavlink
system ID
        mavlink_system.sysid =
g.sysid_this_mav;
        static uint8_t counter;
        counter++;
        // write perf data every 20s
        if (counter % 10 == 0) {
            if (scheduler.debug() != 0) {
                hal.console-
>printf_P(PSTR("G_Dt_max=%lu\n"),
(unsigned long)G_Dt_max);
            }
            if (g.log_bitmask &
MASK_LOG_PM)
                Log_Write_Performance();
```

```
            G_Dt_max = 0;
            resetPerfData();
        }
        // save compass offsets once a minute
        if (counter >= 60) {

            if (g.compass_enabled) {
                compass.save_offsets();
            }
            counter = 0;
        }
    }
    static void update_GPS(void)
    {
        static uint32_t last_gps_reading;
            g_gps->update();

        if (g_gps->last_message_time_ms()
!= last_gps_reading) {
            last_gps_reading = g_gps-
>last_message_time_ms();
            if (g.log_bitmask &
MASK_LOG_GPS) {

DataFlash.Log_Write_GPS(g_gps,
current_loc.alt);
            }
        }
```

```
        have_position =
ahrs.get_projected_position(current_loc);

            if (g_gps->new_data && g_gps-
>status() >= GPS::GPS_OK_FIX_3D) {
                    gps_fix_count++;

                    if(ground_start_count >
1){

            ground_start_count--;

            ground_start_avg += g_gps-
>ground_speed_cm;

                    } else if
(ground_start_count == 1) {
                            // We countdown
N number of good GPS fixes
                            // so that the
altitude is more accurate
                            // ----------------
------------------
                            if
(current_loc.lat == 0) {

            ground_start_count = 5;
```

```
                                } else {
                    init_home();

                    // set system clock for log
timestamps
                    hal.util-
>set_system_clock(g_gps-
>time_epoch_usec());
if (g.compass_enabled) {
            compass.set_initial_location(g_gps
->latitude, g_gps->longitude);
                                            }

            ground_start_count = 0;
                                }
                        }
            ground_speed   = g_gps-
>ground_speed_cm * 0.01;

      #if CAMERA == ENABLED
            if
(camera.update_location(current_loc) ==
true) {
                do_take_picture();
            }
      #endif
            }
      }
```

```
      static void update_current_mode(void)
      {
         switch (control_mode){
         case AUTO:
         case RTL:
         case GUIDED:
            calc_lateral_acceleration();
            calc_nav_steer();
            calc_throttle(g.speed_cruise);
            break;

         case STEERING: {
            /*
               in steering mode we control
lateral acceleration
               directly. We first calculate the
maximum lateral
               acceleration at full steering lock
for this speed. That is
               V^2/R where R is the radius of
turn. We get the radius of
               turn from half the
STEER2SRV_P.
               */
               float max_g_force = ground_speed
* ground_speed /
steerController.get_turn_radius();
```

```
            // constrain to user set
TURN_MAX_G
            max_g_force =
constrain_float(max_g_force, 0.1f,
g.turn_max_g * GRAVITY_MSS);
            lateral_acceleration = max_g_force
* (channel_steer-
>pwm_to_angle()/4500.0f);
            calc_nav_steer();

            // and throttle gives speed in
proportion to cruise speed, up
            // to 50% throttle, then uses
nudging above that.
            float target_speed =
channel_throttle->pwm_to_angle() * 0.01 *
2 * g.speed_cruise;
            in_reverse = (target_speed < 0);
            if (in_reverse) {
               target_speed =
constrain_float(target_speed, -
g.speed_cruise, 0);
            } else {
               target_speed =
constrain_float(target_speed, 0,
g.speed_cruise);
            }
            calc_throttle(target_speed);
            break;
         }

         case LEARNING:
         case MANUAL:
            /*
              in both MANUAL and
LEARNING we pass through the
              controls. Setting servo_out here
actually doesn't matter, as
              we set the exact value in
set_servos(), but it helps for
              logging
             */
            channel_throttle->servo_out =
channel_throttle->control_in;
            channel_steer->servo_out =
channel_steer->pwm_to_angle();

            // mark us as in_reverse when
using a negative throttle to
            // stop AHRS getting off
            in_reverse = (channel_throttle-
>servo_out < 0);
            break;
         case HOLD:
```

```
            // hold position - stop motors and
center steering
            channel_throttle->servo_out = 0;
            channel_steer->servo_out = 0;
            break;
        case INITIALISING:
            break;
            }
    }

    static void update_navigation()
    {
        switch (control_mode) {
        case MANUAL:
        case HOLD:
        case LEARNING:
        case STEERING:
        case INITIALISING:
            break;
```

```
        case AUTO:
                verify_commands();
            break;

        case RTL:
        case GUIDED:
            // no loitering around the wp with
the rover, goes direct to the wp position
            calc_lateral_acceleration();
            calc_nav_steer();
            if (verify_RTL()) {
                channel_throttle->servo_out =
g.throttle_min.get();
                set_mode(HOLD);
            }
            break;
            }
      }

AP_HAL_MAIN();
```