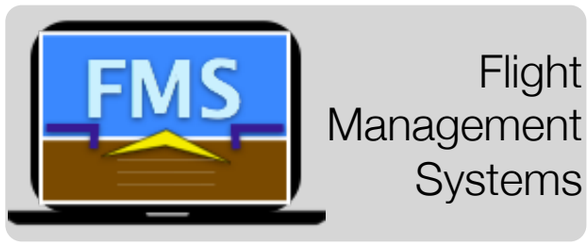


Flight Management Systems

Flight simulator
Writing a simple autopilot

Joan Vila & Ángel Rodas

Building a simple autopilot

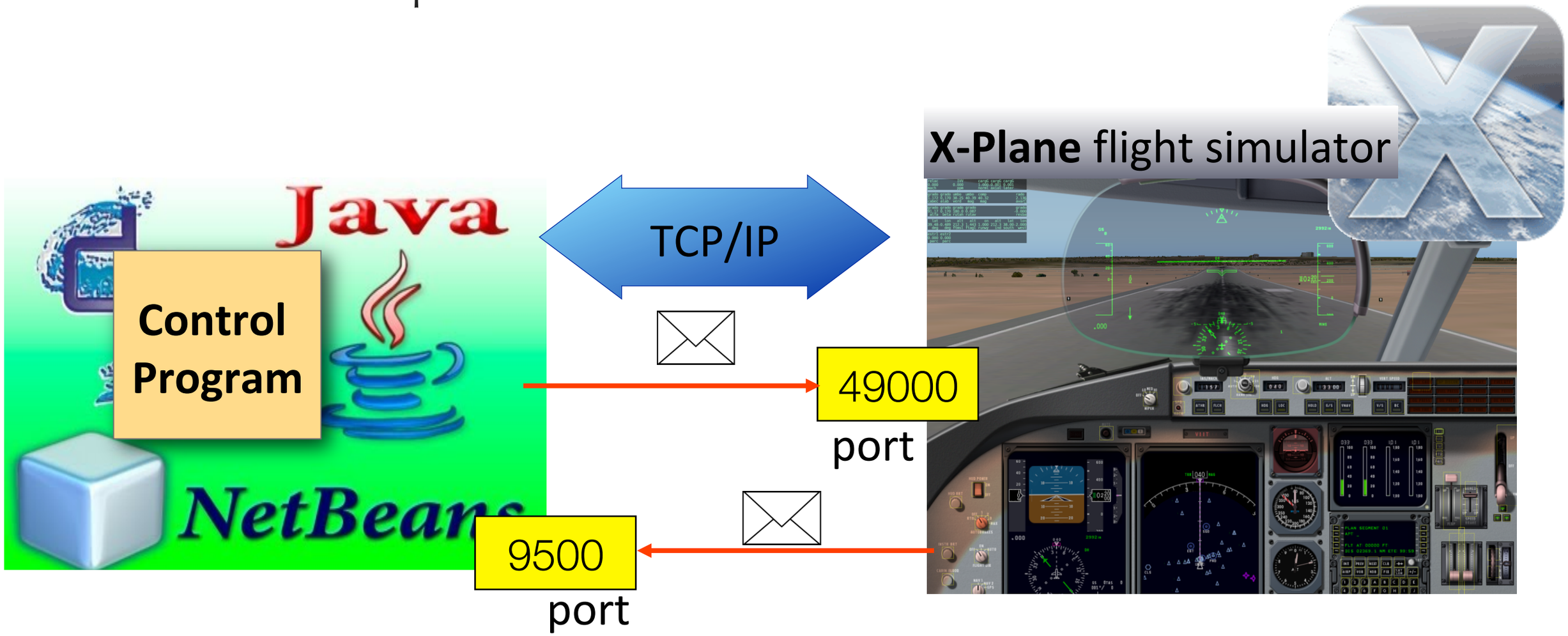
Flight Management Systems

- **Communicating Java and XPlane using UDP**
- **A simple autopilot**

Communicating Java and XPlane

Flight Management Systems

- **UDP communication**
  - Communication ports



Communicating Java and XPlane

Flight Management Systems

- **The “datarefs” tree**

Communicating Java and XPlane

- **The “datarefs” tree**
  - http://www.xsquawkbox.net/xpsdk/docs/DataRefs.html

+ sim/flightmodel/position/

latitude	double	660+	no	degrees	The latitude of the aircraft
longitude	double	660+	no	degrees	The longitude of the aircraft
elevation	double	660+	no	meters	The elevation above MSL of the aircraft
theta	float	660+	yes	degrees	The pitch relative to the plane normal to the Y axis in degrees - OpenGL coordinates
phi	float	660+	yes	degrees	The roll of the aircraft in degrees - OpenGL coordinates
psi	float	660+	yes	degrees	The true heading of the aircraft in degrees from the Z axis - OpenGL coordinates

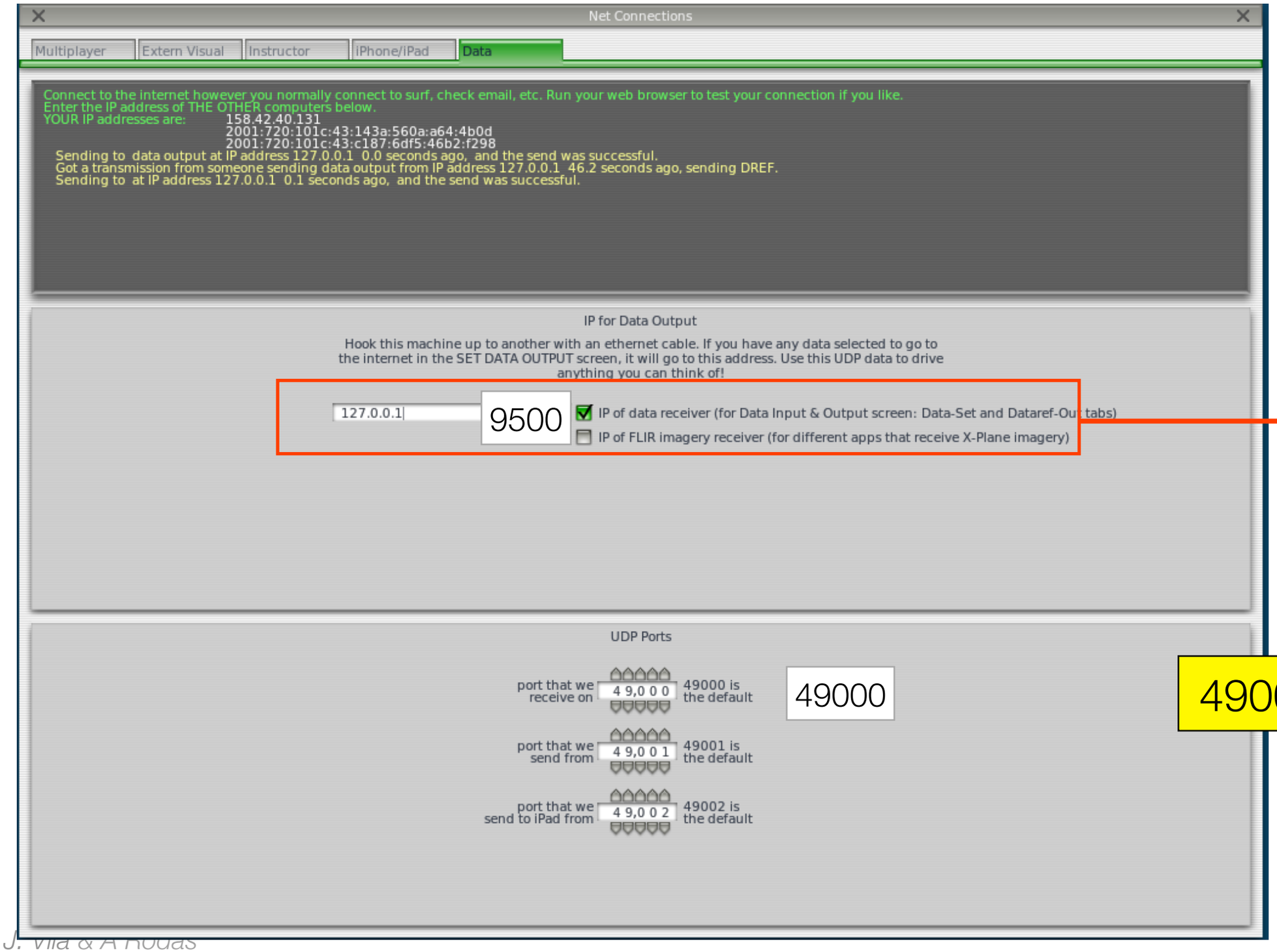
+ sim/joystick/

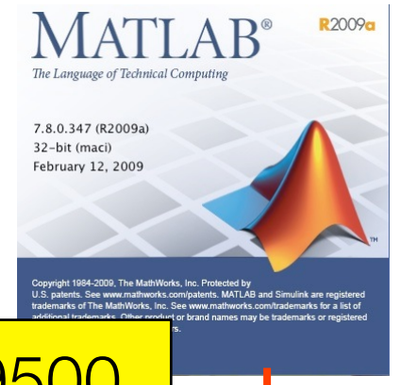
yoke_pitch_ratio	float	900+	yes	[-1..1]	The deflection of the joystick axis controlling pitch. Use override_joystick or override_joystick_pitch
yolk_pitch_ratio	float	660+	yes	[-1..1]	Legacy - this spelling error is still present for
yoke_roll_ratio	float	900+	yes	[-1..1]	The deflection of the joystick axis controlling roll. Use override_joystick or override_joystick_roll
yolk_roll_ratio	float	660+	yes	[-1..1]	Legacy - this spelling error is still present for backward compatibility with older plugins.
yoke_heading_ratio	float	900+	yes	[-1..1]	The deflection of the joystick axis controlling yaw. Use override_joystick or override_joystick_heading

Communicating Java and XPlane

Flight Management Systems

- **Change the X-plane net configuration**





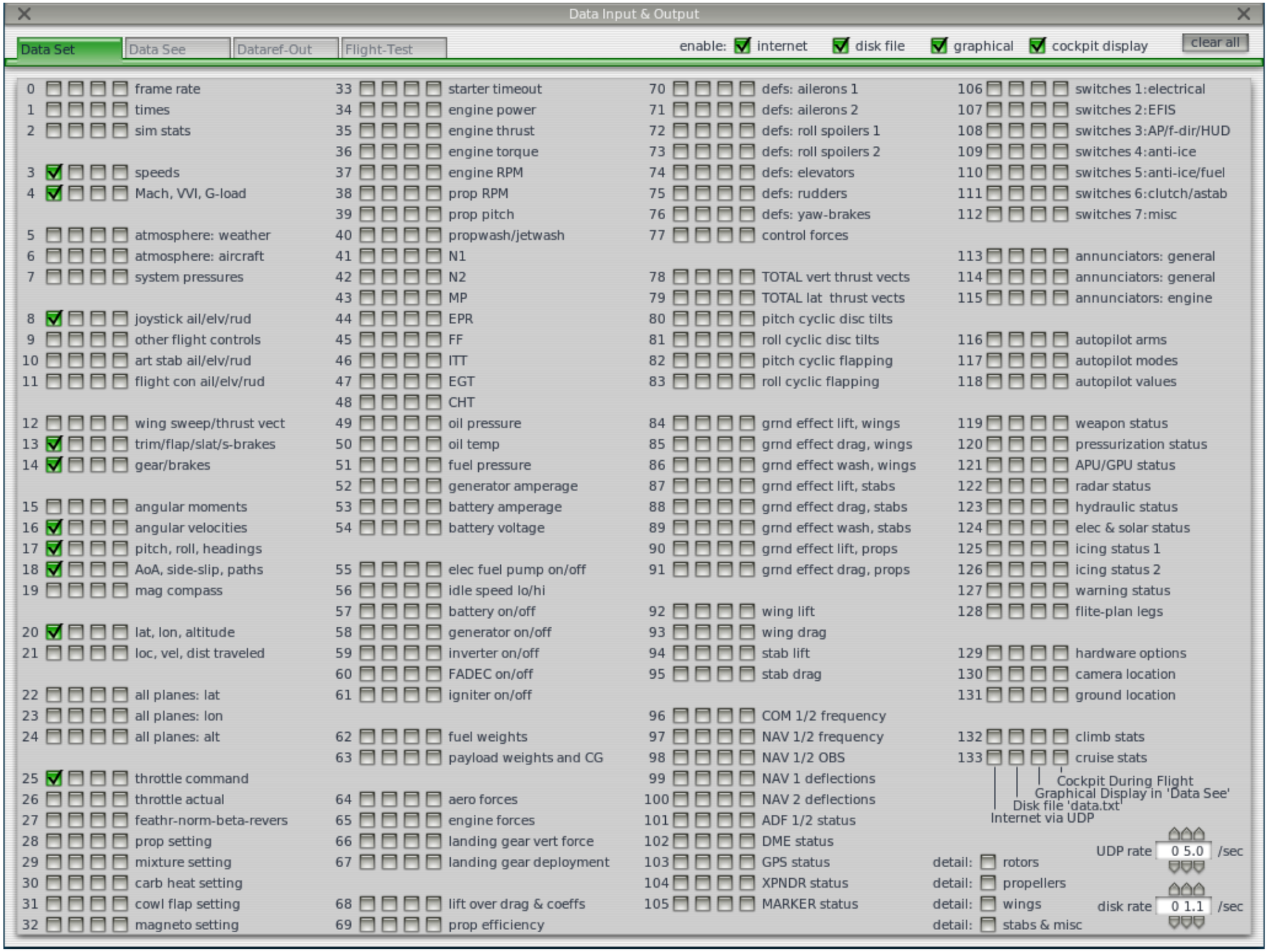
9500

49000

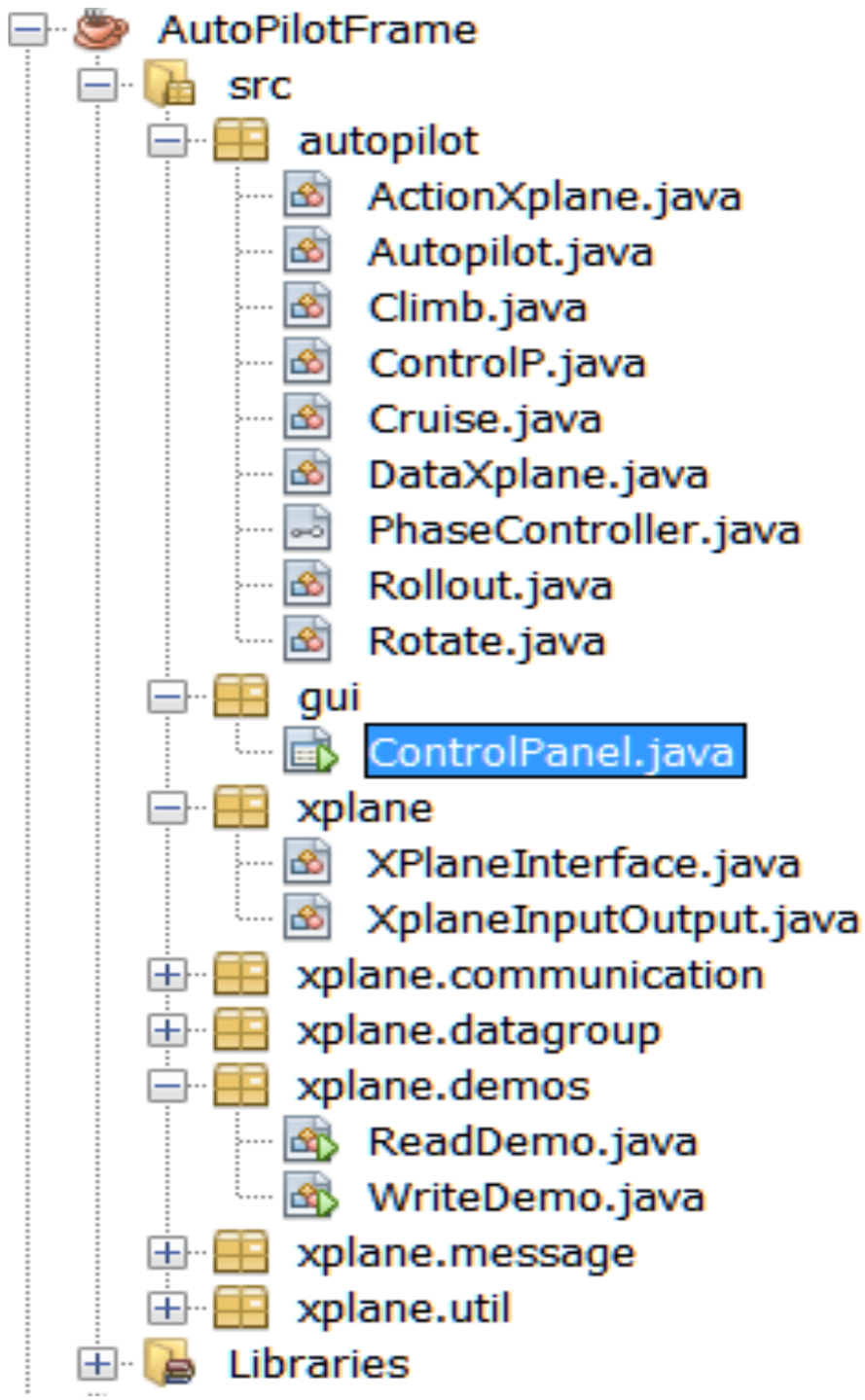
Communicating Java and XPlane

Flight Management Systems

- **X-plane data output configuration**



Java Project



DataXplane and ActionXplane classes

Flight Management Systems

```java
public class ActionXplane {

    private int phase; // current phase
    private float ailerons;
    private float elevators;
    private float rudder;
    private float throttle;
    private float brake;
    private float flaps;

    public ActionXplane() {
        ailerons = 0;
        elevators = 0;
        rudder = 0;
        throttle = 0;
        brake = 1;
        flaps = 0;
    }

    public void setPhase(int phase) {
        this.phase = phase;
    }
```

```java
public class DataXplane {

    private float lat;
    private float lon;
    private float alt;
    private float roll;
    private float pitch;
    private float head;

    private float aoa;
    private float beta;
    private float vpath;
    private float hpath;

    private float ias;
    private float tas;
    private float gs;
    private float vs;

    private float brake;

    public void setLat(float lat) {
        this.lat = lat;
    }
```

XplaneInputOutput class

Flight Management Systems

```
public class XplaneInputOutput {

    private XPlaneInterface xpi;
    private DataXplane data;

    public XplaneInputOutput() {

        try {
            xpi = new XPlaneInterface("127.0.0.1", 9500, 49000, "DATAGroupConfig.xml");
            xpi.unregisterDATAMessages("*");
            xpi.registerDATAMessages("3,4,8,13,14,16,17,18,20,25"); // 20 lat lon alt replaces  22 23 24
            xpi.startReceiving();
            try {
                Thread.sleep(1000); // wait a few before send actions
            } catch (InterruptedException ex) {
                Logger.getLogger(ControlPanel.class.getName()).log(Level.SEVERE, null, ex);
            }
        } catch (SocketException ex) {
            Logger.getLogger(ControlPanel.class.getName()).log(Level.SEVERE, null, ex);
        } catch (UnknownHostException ex) {
            Logger.getLogger(ControlPanel.class.getName()).log(Level.SEVERE, null, ex);
        }
        data = new DataXplane();

    }
```

XplaneInputOutput class

Flight Management Systems

```
public DataXplane read() {

    data.setLat(xpi.getValue("position.lat"));
    data.setLon(xpi.getValue("position.lon"));
    data.setAlt(xpi.getValue("position.altMsl"));
    data.setRoll(xpi.getValue("orientation.roll"));
    data.setPitch(xpi.getValue("orientation.pitch"));
    data.setHead(xpi.getValue("orientation.headingTrue"));
    data.setAoa(xpi.getValue("orientation.alfa"));
    data.setBeta(xpi.getValue("orientation.beta"));
    data.setVpath(xpi.getValue("orientation.vpath"));
    data.setHpath(xpi.getValue("orientation.hpath"));
    data.setBrake(xpi.getValue("controls.brake"));
    data.setIas(xpi.getValue("position.ias"));
    data.setTas(xpi.getValue("position.tas"));
    data.setGs(xpi.getValue("position.gs"));
    data.setVs(xpi.getValue("position.vs"));
    return data;

  }

public void write(ActionXplane action) {

    xpi.setValue("controls.aileronsPosition", action.getAilerons());
    xpi.setValue("controls.elevatorsPosition", action.getElevators());
    xpi.setValue("controls.rudderPosition", action.getRudder());
    xpi.setValue("engine.throttleCommand1", action.getThrottle());
    xpi.setValue("controls.brake", action.getBrake());
    xpi.setValue("controls.flapHandl", action.getFlaps());

}
```

Demos. Reading data from X-Plane

Flight Management Systems

```java
public class ReadDemo {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        XplaneInputOutput xplane;
        DataXplane data;

        xplane = new XplaneInputOutput();

        try {
            Thread.sleep(1000); // wait a little before reading
        } catch (InterruptedException ex) {
        }

        data = xplane.read();
        System.out.println("Latitude " + data.getLat() + " Longitude " + data.getLon() + " Altitude " + data.getAlt());

        xplane.stop();

    }

}
```

Demos. Writing data to X-Plane

Flight Management Systems

```
public class WriteDemo {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        XplaneInputOutput xplane;
        ActionXplane action;

        xplane = new XplaneInputOutput();
        action = new ActionXplane();
        action.setAilerons(-1);
        action.setElevators(0);
        action.setRudder(-1);
        action.setThrottle(1);
        action.setBrake(0);
        action.setFlaps(1);
        xplane.write(action);

        xplane.stop();


    }

}
```

Building a simple autopilot

Flight Management Systems

- **Communicating Java and XPlane using UDP**
- **A simple autopilot**

Autopilot class


```java
public class Autopilot extends Thread {

    public static final int ROLLOUT = 1;
    public static final int ROTATE = 2;
    public static final int CLIMB = 3;
    public static final int CRUISE = 4;

    XplaneInputOutput xplane;
    ControlPanel mygui;
    private boolean running;
    private int sampleRate;

    private DataXplane data;
    private ActionXplane action;

    private PhaseController pController;
    private Rollout rollOut;
    private Rotate rotate;
    private Climb climb;
    private Cruise cruise;

    public Autopilot(ControlPanel mygui) {

        this.xplane = new XplaneInputOutput();
        this.mygui = mygui;

        try {
            Thread.sleep(1000); // wait a few before
        } catch (InterruptedException ex) {
            Logger.getLogger(ControlPanel.class.getNa
        }
        rollOut = new Rollout(); // inicialize phases
        rotate = new Rotate();
        climb = new Climb();
        cruise = new Cruise();

        running = true;
        sampleRate = 200;
        this.start();
    }
```

```java
    @Override
    public void run() {

        pController = rollOut; // initial phase

        while (running) { // control loop

            data = xplane.read();
            mygui.display(data);
            action = pController.computeAction(data);
            xplane.write(action);
            mygui.display(action);

            switch (action.getPhase()) {
                case ROLLOUT:
                    pController = rollOut;
                    break;
                case ROTATE:
                    pController = rotate;
                    break;
                case CLIMB:
                    pController = climb;
                    break;
                case CRUISE:
                    pController = cruise;
                    break;
            }

            try {
                Thread.sleep(sampleRate);
            } catch (InterruptedException ex) {
                Logger.getLogger(ControlPanel.class.get
            }
        }

        xplane.stop();
    }
```

PhaseController interface and ControlP


```
public class ControlP {

    float Kp;
    float minv;
    float maxv;

    public ControlP(float Kp, float minv, float maxv) {
        this.Kp = Kp;
        this.minv = minv;
        this.maxv = maxv;
    }

    public float control(float err) {
        float action;
        action = Kp * err;
        action = Math.min(maxv, action);
        action = Math.max(minv, action);
        return action;
    }

}
```

PhaseController interface and ControlP

Flight Management Systems

```java
public interface PhaseController {

    public ActionXplane computeAction(DataXplane data);

}
```

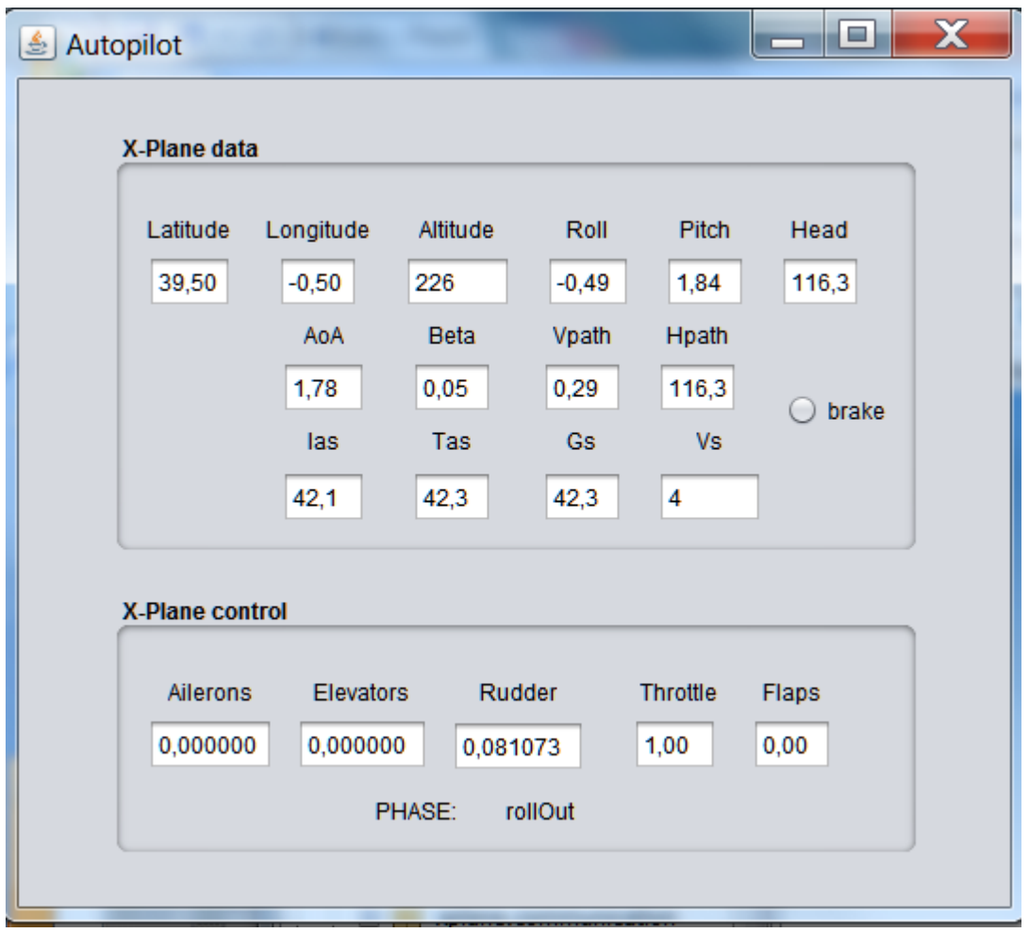
```java
public class Rollout implements PhaseController {

    private ActionXplane actions;
    private ControlP rudder_control;

    public Rollout() {
        actions = new ActionXplane();
        rudder_control = new ControlP(0.03f, -1, 1);

    }

    @Override
    public ActionXplane computeAction(DataXplane data) {
```

```java
public class Rotate implements PhaseController {

    private ActionXplane actions;
    private ControlP lateral_guidance;
    private ControlP lateral_control;

    public Rotate() {
        actions = new ActionXplane();
        lateral_guidance = new ControlP(0.2f, -20, 20);
        lateral_control = new ControlP(0.05f, -1, 1);
    }

    @Override
    public ActionXplane computeAction(DataXplane data) {
```

```java
public class Climb implements PhaseController {

    private ActionXplane actions;
    private ControlP lateral_guidance;
    private ControlP lateral_control;
    private ControlP vertical_guidance;
    private ControlP vertical_control;

    public Climb() {
        actions = new ActionXplane();
        lateral_guidance = new ControlP(0.2f, -20, 20);
        lateral_control = new ControlP(0.05f, -1, 1);
        vertical_guidance = new ControlP(1, -15, 15);
        vertical_control = new ControlP(0.001f, -1, 1);
    }

    @Override
    public ActionXplane computeAction(DataXplane data) {
```

```java
public class Cruise implements PhaseController {

    private ActionXplane actions;
    private ControlP lateral_guidance;
    private ControlP lateral_control;
    private ControlP vertical_guidance;
    private ControlP vertical_control;

    public Cruise() {
        actions = new ActionXplane();
        lateral_guidance = new ControlP(0.2f, -20, 20);
        lateral_control = new ControlP(0.05f, -1, 1);
        vertical_guidance = new ControlP(0.6f, -15, 15);
        vertical_control = new ControlP(0.006f, -1, 1);
    }

    @Override
    public ActionXplane computeAction(DataXplane data) {
```

GUI. ControlPanel class




```
public class ControlPanel extends javax.swing.JFrame {

    /**
     * Creates new form ControlPanel
     */
    private Autopilot aPilot;

    public ControlPanel() {
        initComponents();
        aPilot=new Autopilot(this);
    }

    public void display(DataXplane dx) {

        latText.setText(String.format("%.2f", dx.getLat()));
        lonText.setText(String.format("%.2f", dx.getLon()));
        altText.setText(String.format("%.0f", dx.getAlt()));
        rollText.setText(String.format("%.2f", dx.getRoll()));
            ...

    public void display(ActionXplane ax) {

        String phase="";
        switch (ax.getPhase()) {
            case Autopilot.ROLLOUT:
                phase = "rollOut";
                break;
            case Autopilot.ROTATE:
                phase = "rotate";
                break;
            case Autopilot.CLIMB:
                phase = "climb";
                break;
            case Autopilot.CRUISE:
                phase = "cruise";
                break;
        }
        phaseLabel.setText(phase);
        ailText.setText(String.format("%.6f", ax.getAilerons()));
        eleText.setText(String.format("%.6f", ax.getElevators()));
        rudText.setText(String.format("%.6f", ax.getRudder()));
        thrText.setText(String.format("%.2f", ax.getThrottle()));
        brakeRadioButton.setSelected(ax.getBrake() != 0);
        flapText.setText(String.format("%.2f", ax.getFlaps()));

    }
```

Exercise: a simple autopilot

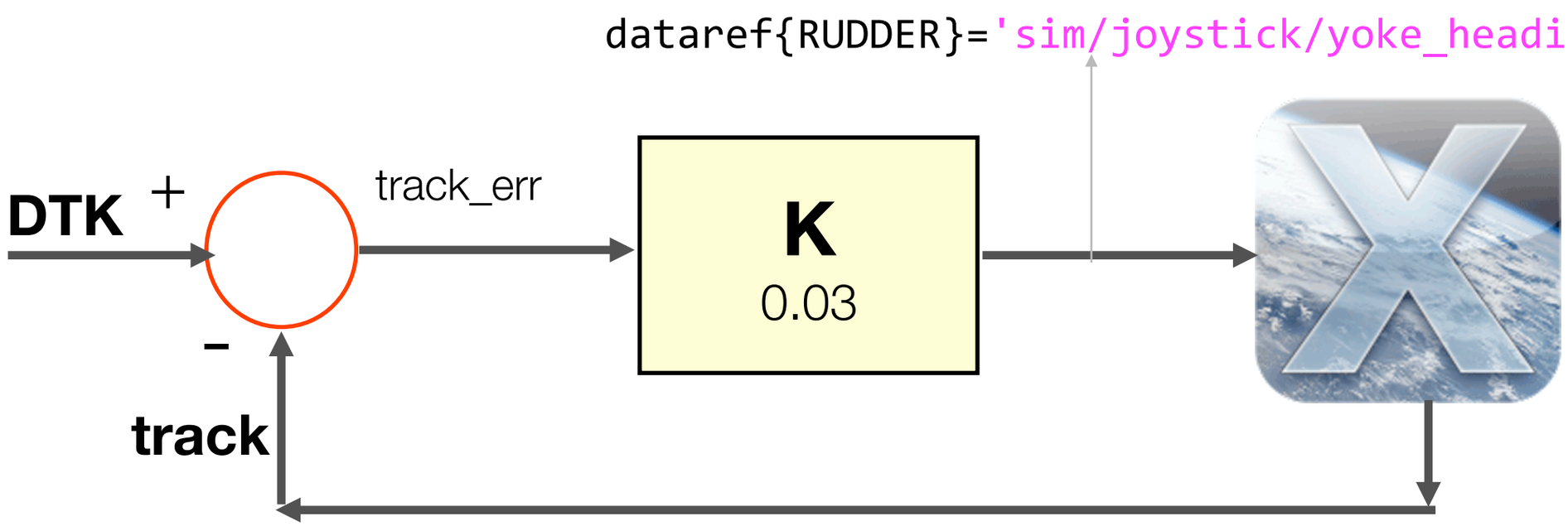
- **A simple autopilot for the Cessna 172**
  - Complete the Netbeans project to take off the Cessna from LEVC RWY 12 runway and take it cruise phase to 2000 ft and heading 120.
  - The program will consist of the following phases:
    - *Rollout*
    - *Rotation*
    - *Climb*
    - *Cruise*

Exercise: a simple autopilot

- **Rollout phase**
  - **Throttle**: full
  - **Elevators**: no deflection
  - **Rudder control**: the rudder controls the heading on ground because it acts on the front wheel.
    - ***Control policy**: Proportional control. Based on track error.*
      - DTK = 120;
      - error_h = DTK-track;
      - actions(RUDDER) = Krud * error_h;
    - *Try with different Krud. Suggestion: Krud = 0.03*
  - **Ailerons**: no deflection
  - **Transition** to the next phase when the TAS > 70 kts.

Exercise: a simple autopilot

- **Rollout phase**
  - **Rudder control**



Rollout code


```java
public class Rollout implements PhaseController {

    private ActionXplane actions;
    private ControlP rudder_control;

    public Rollout() {
        actions = new ActionXplane();
        rudder_control = new ControlP(0.03f, -1, 1);

    }
```

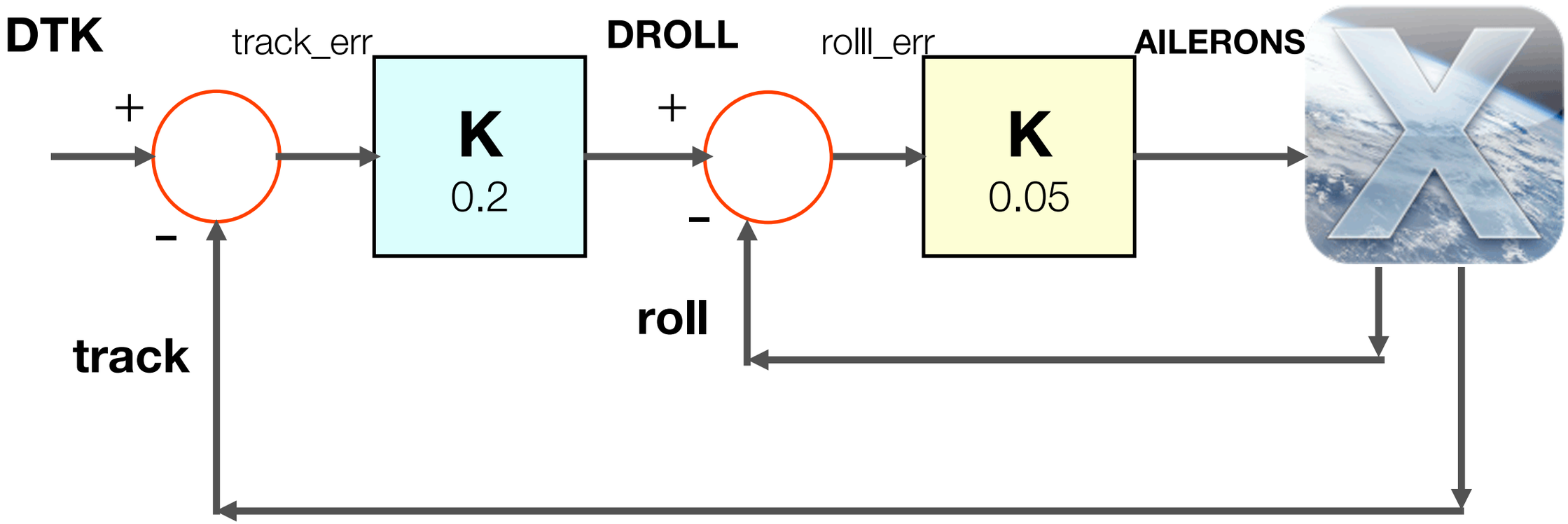
```java
@Override
public ActionXplane computeAction(DataXplane data) {

    final float DTK = 119; // references
    final float TAS_MAX = 70; // termination conditions
    float tas, track; // inputs
    float rudder; // outputs
    float track_error; // errors

    // READ DATA-----------------------------------------
    tas = data.getTas();
    track = data.getHpath();

    // LATERAL GUIDANCE----------------------------------
    // CONTROL LOOP
    track_error = DTK - track;
    rudder = rudder_control.control(track_error);

    // SET ACTIONS---------------------------------------
    actions.setAilerons(0);
    actions.setElevators(0);
    actions.setRudder(rudder);
    actions.setThrottle(1);
    actions.setBrake(0);
    actions.setFlaps(0.0f);

    // SET NEXT FLIGHT PHASE-----------------------------
    if (tas > TAS_MAX) {
        actions.setPhase(Autopilot.ROTATE);
    } else {
        actions.setPhase(Autopilot.ROLLOUT);
    }

    return actions;
}
```

Exercise: a simple autopilot

- **Rotate phase**
  - **Throttle**: full
  - **Elevators**: a fix value to start climbing.
    - *Suggestion: 0.01*
  - **Rudder**: navigating straight does not requiere to use rudder actions.
  - **Ailerons**: ailerons are used to control the roll and thus the track angles.
    - ***Control policy**: Proportional control. Double loop:*
    - ***Guidance loop**: compute roll angle based on track error. K=0.2*
    - ***Control loop**: control roll angle based on roll error. K=0.05*
- **Transition** to the next phase when altitude > 400 ft.

Exercise: a simple autopilot

- **Rotate phase**
  - **Ailerons control**



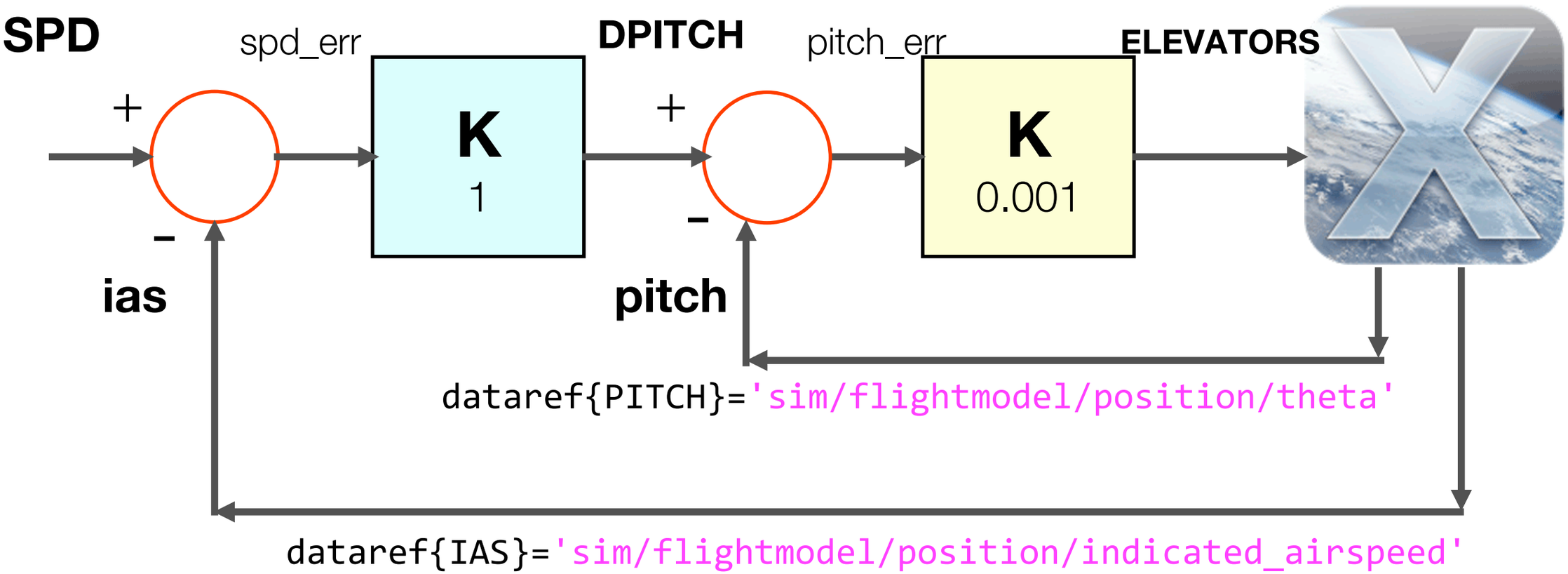
Exercise: a simple autopilot

- **Climb phase**
  - **Throttle**: full
  - **Elevators**: Climb after takes off is a constant speed climb (TAS/IAS). The target value for (TAS/IAS) will be 90 kts.
    - ***Elevators control law**: if the speed decreases, the pitch angle will be reduced to increase speed and avoid stall and viceversa.*
      - ***Control policy**: Proportional control. Double loop:*
      - ***Guidance loop**: compute pitch angle based on speed error. K=1*
      - ***Control loop**: control pitch angle based on pitch error. K=0.001*
  - **Rudder**: navigating in straight does not requiere to use rudder actions.
  - **Ailerons**: Similar to previous phase.
  - **Transition** to the next phase when altitude > 2000 ft.

Exercise: a simple autopilot

Flight Management Systems

- **Climb phase**
  - **Elevators control**



Exercise: a simple autopilot

- **Cruise phase**
  - The goal of this phase will be keeping a **constant altitude of 2000** ft and turning to **course 060º**.
  - **Throttle**: reduce to 50%. Watch the tachometer is in the green area to adjust the right value.
  - **Elevators**: keep constant altitude of 2000 ft. Perform a proportional control based on altitude error.
    - ***Control policy**: Proportional control. Double loop:*
    - ***Guidance loop**: compute pitch angle based on altitude error. K=0.6*
    - ***Control loop**: control pitch angle based on pitch error. K=0.006*
  - **Rudder**: no deflection.
  - **Ailerons**: Similar to previous phase. Now the DTK is 060º.
  - **Transition:** None.