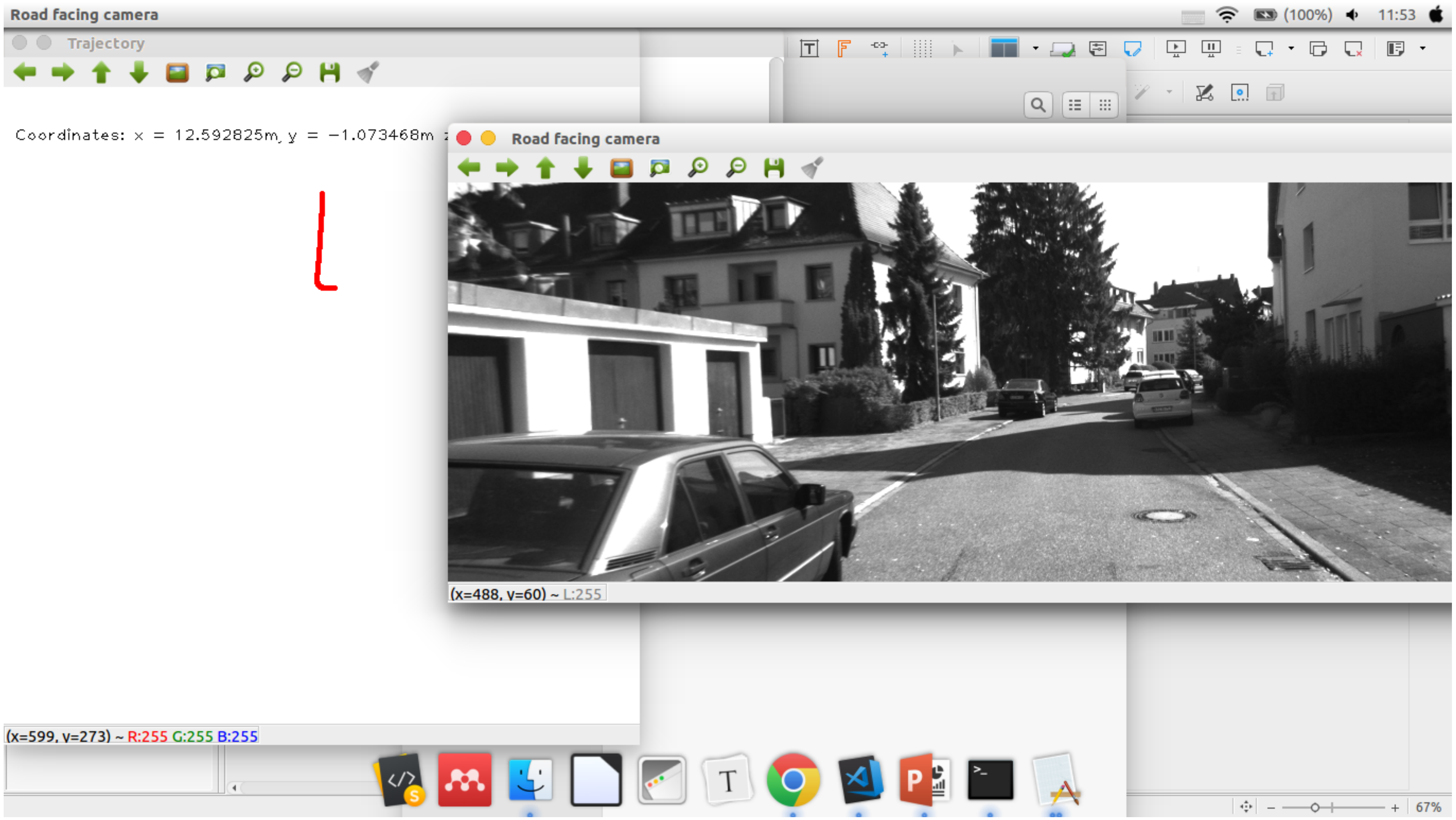
MonoVo

李建峰　冯亚炫

效果展示


Road facing camera

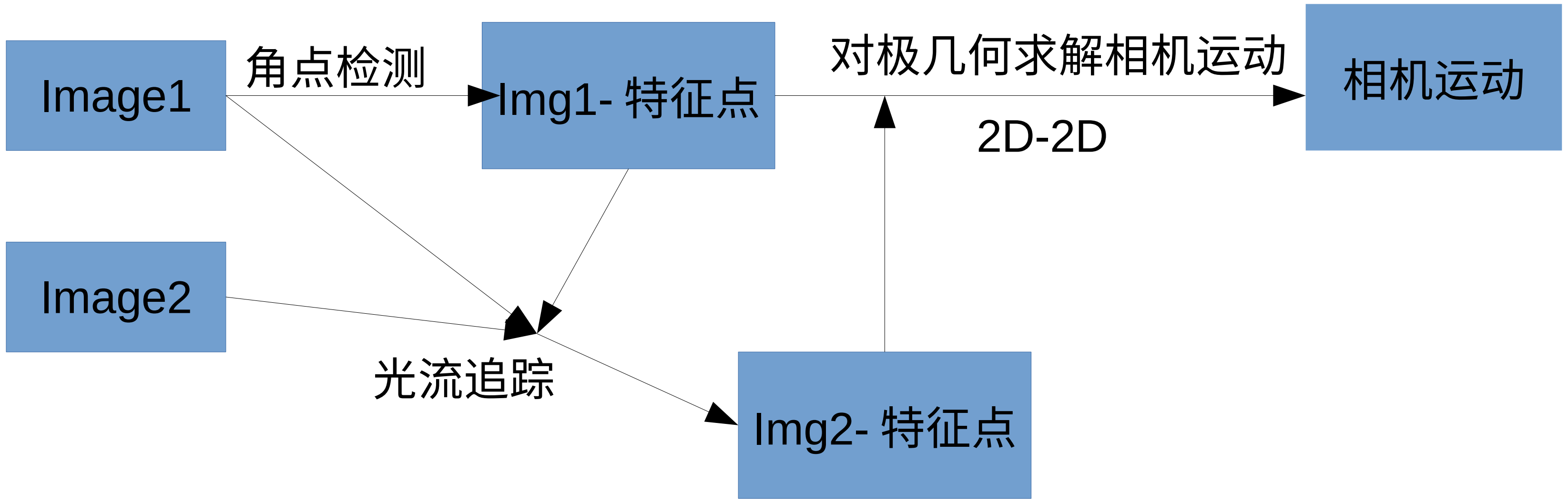
Trajectory

```
Coordinates: x = 12.592825m, y = -1.073468m
```

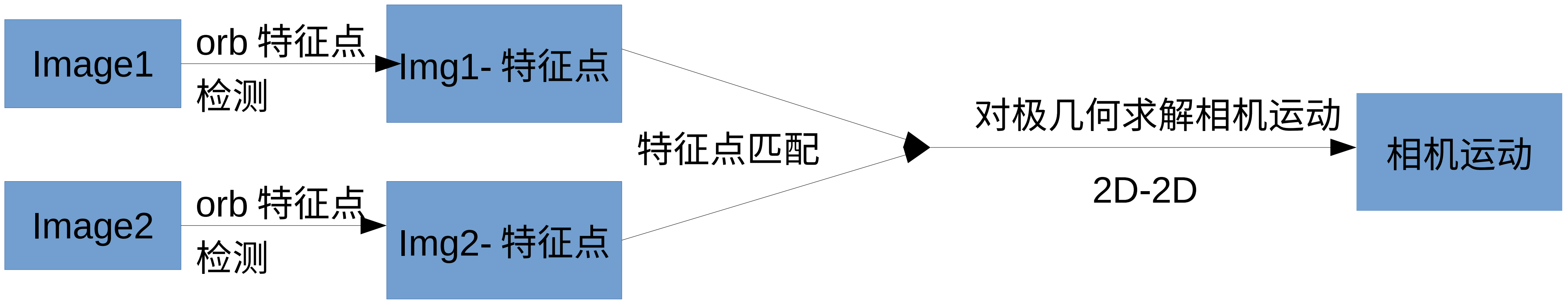
(x=599, y=273) ~ R:255 G:255 B:255

(x=488, y=60) ~ L:255

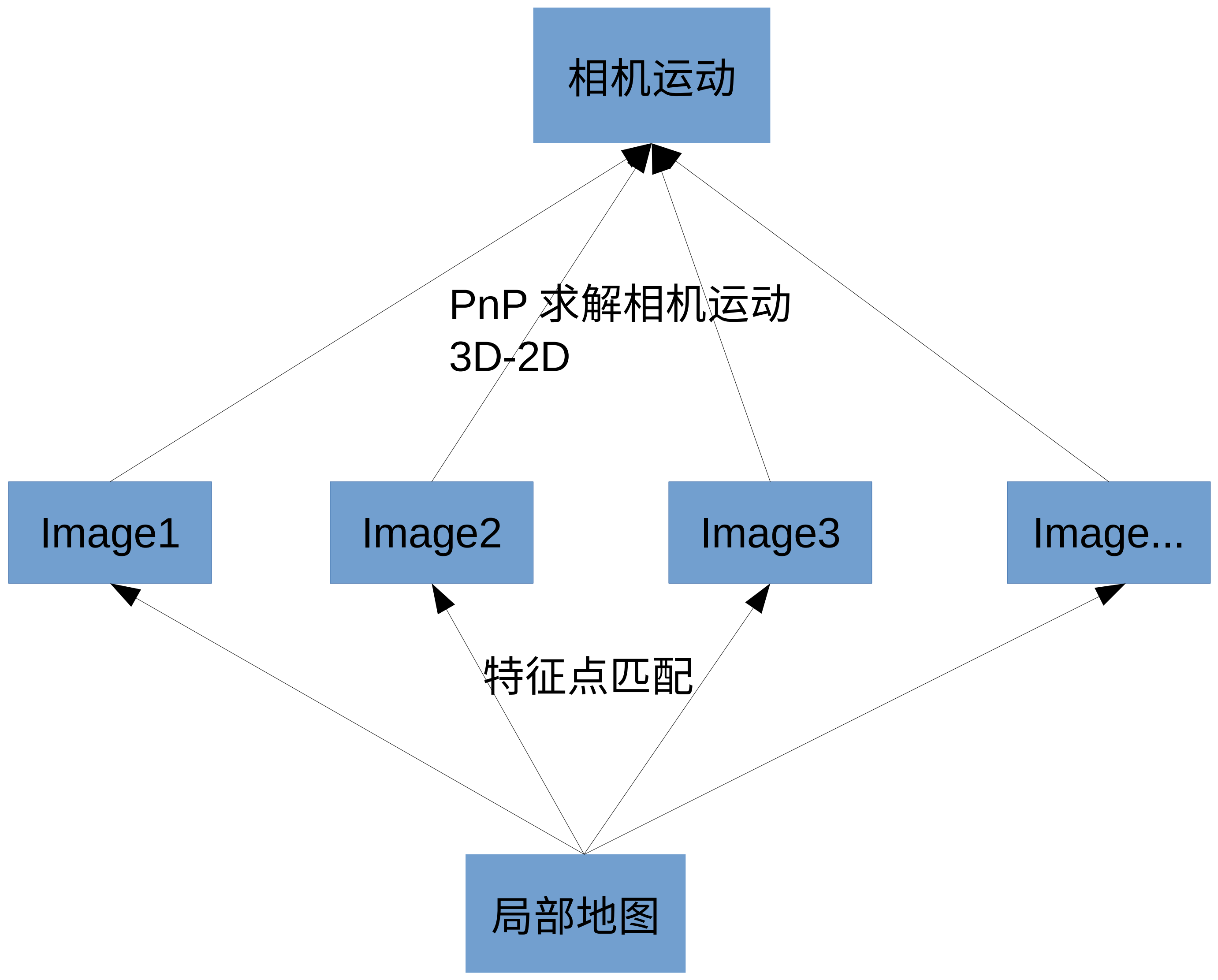
基于光流追踪的视觉里程计



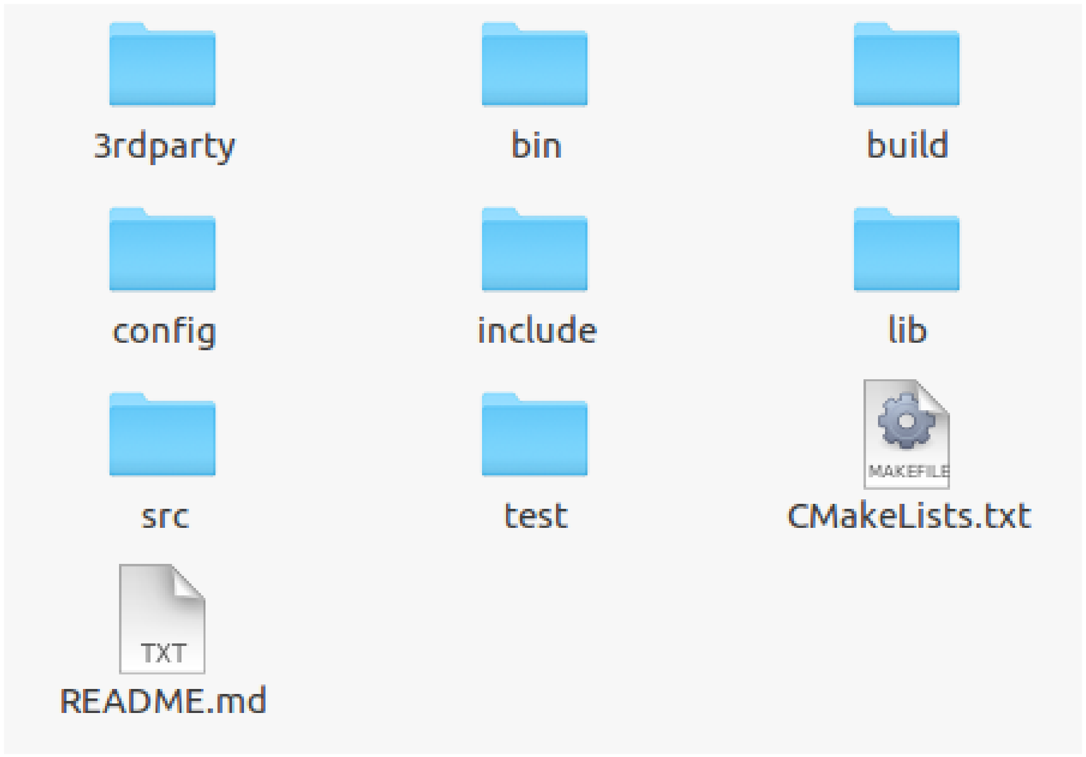
基于特征点匹配的视觉里程计

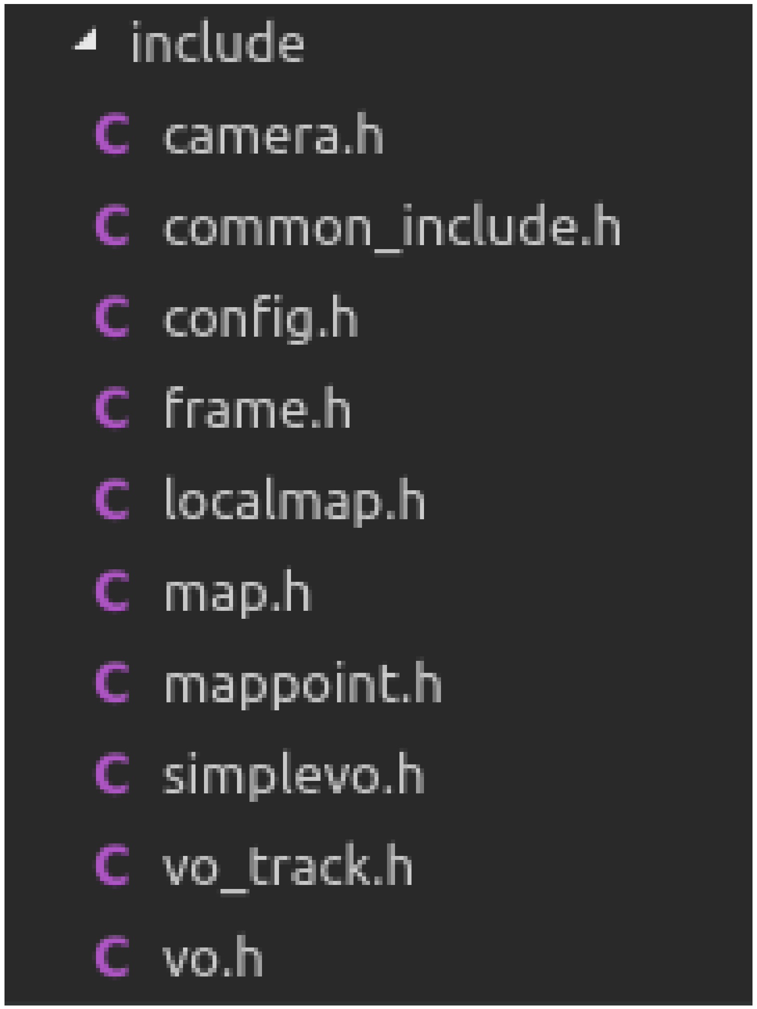


基于匹配附加局部地图的视觉里程计

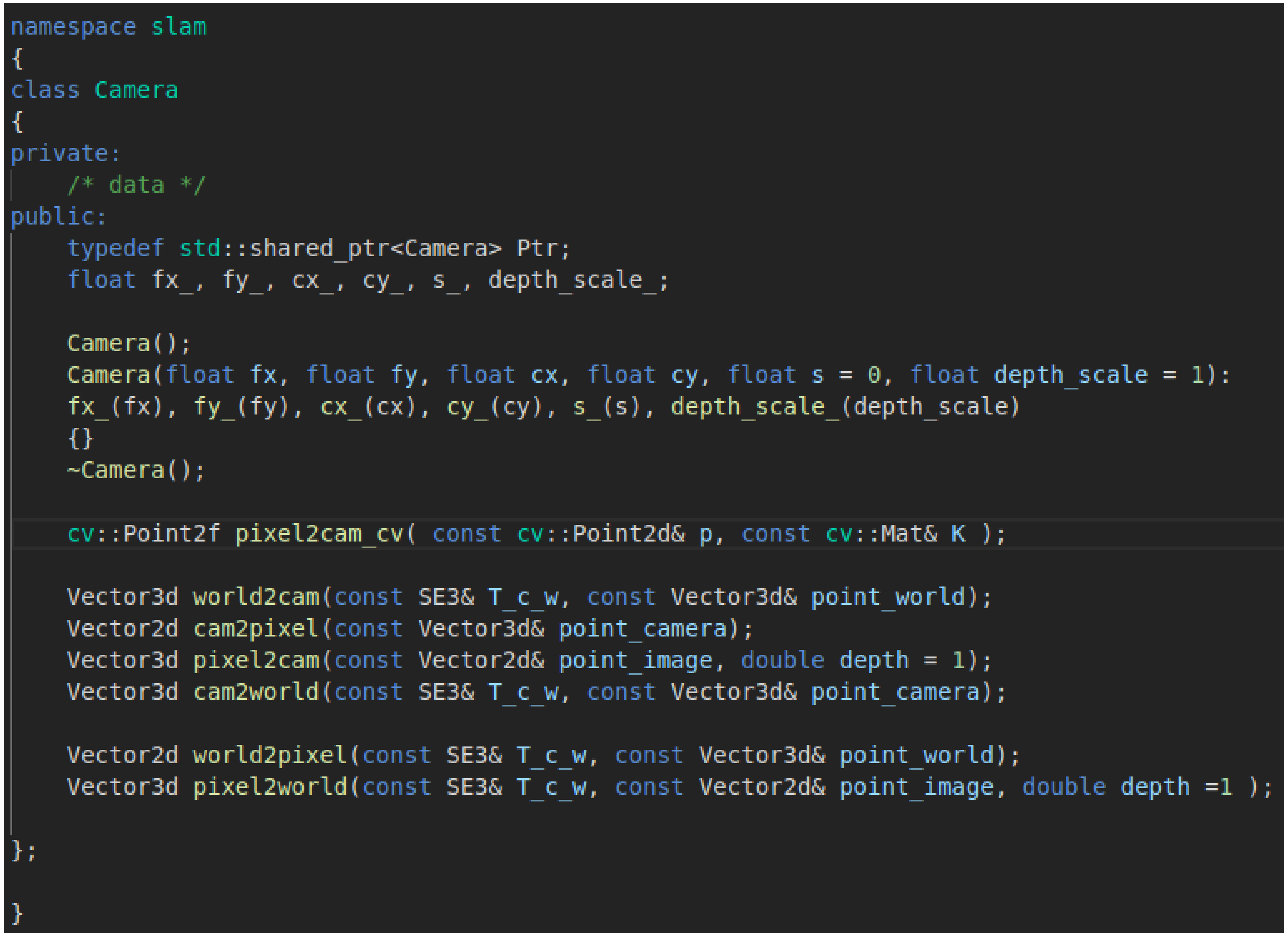


代码实现

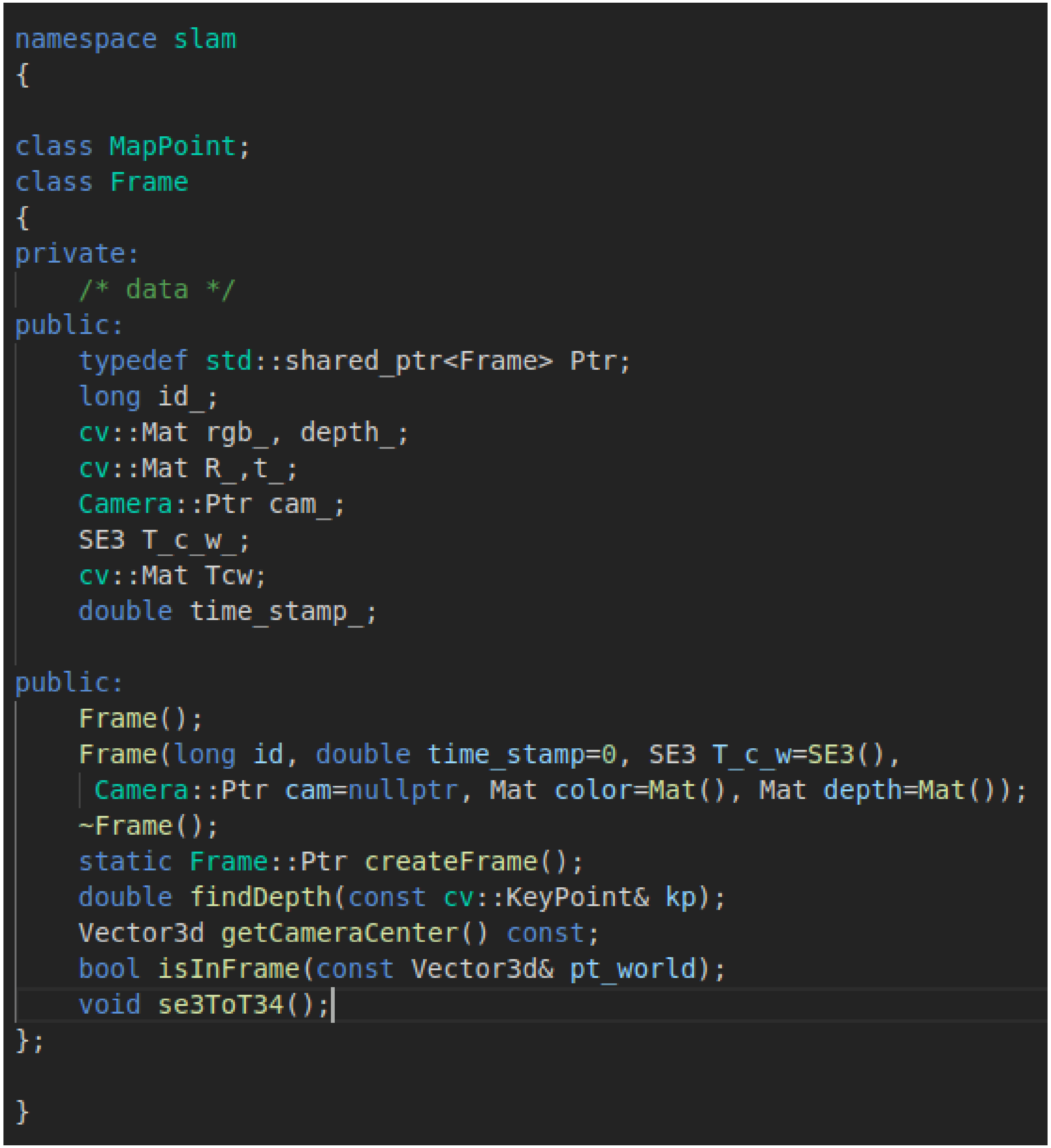

3rdparty
bin
build
config
include
lib
src
test
CMakeLists.txt
README.md

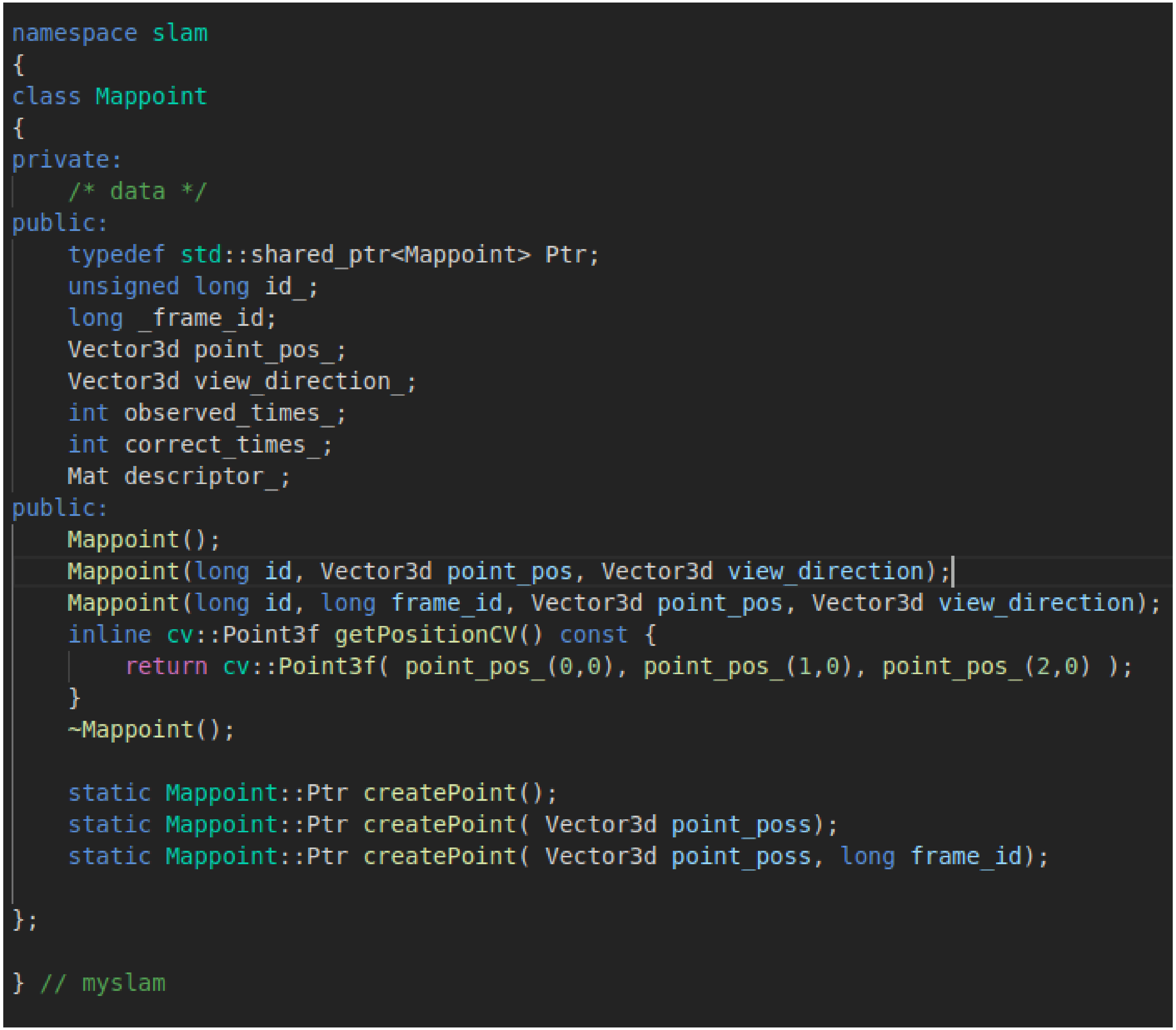

include
camera.h
common_include.h
config.h
frame.h
localmap.h
map.h
mappoint.h
simplevo.h
vo_track.h
vo.h

代码实现—类的组织与功能

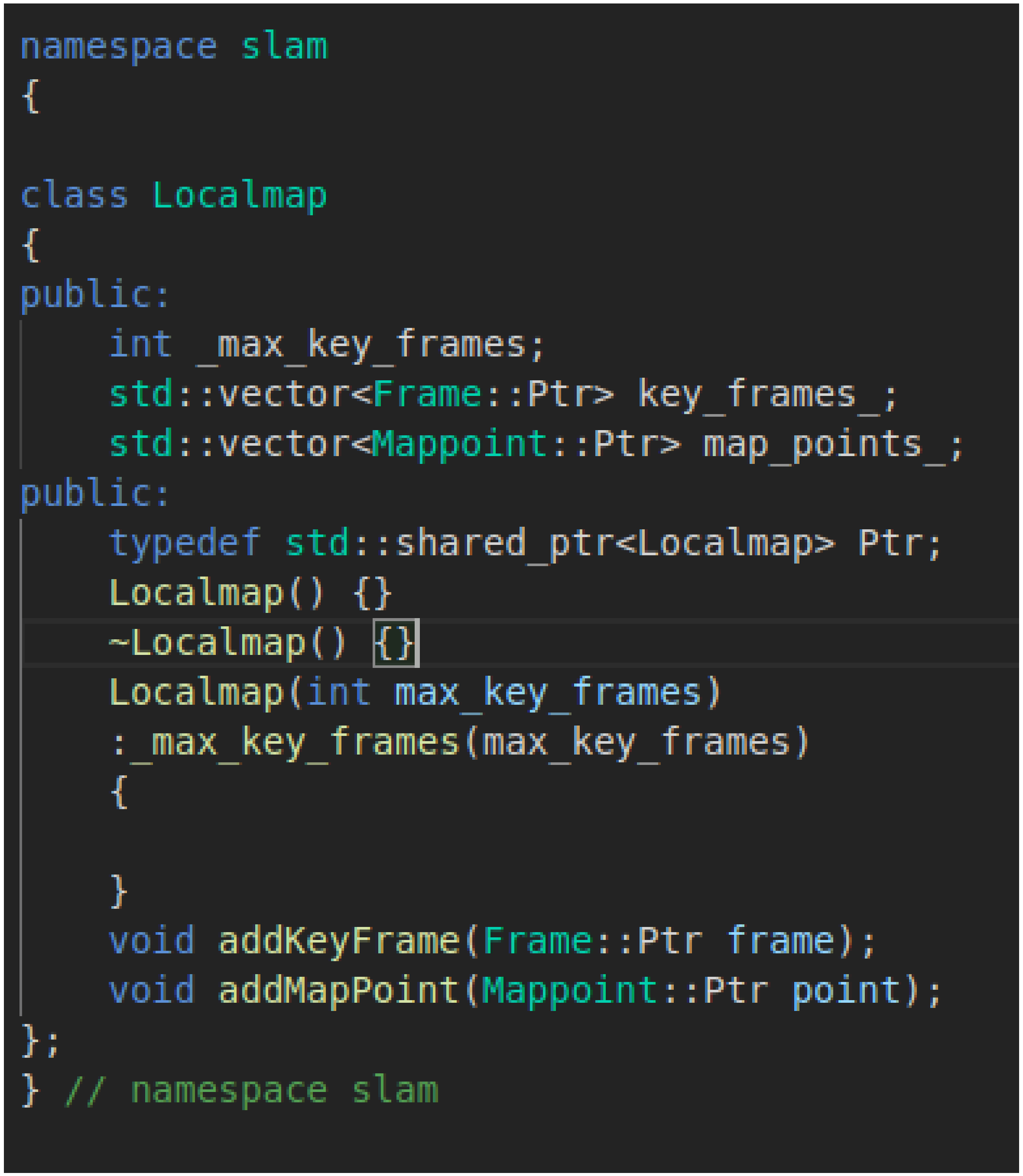


```cpp
namespace slam
{
class Camera
{
private:
    /* data */
public:
    typedef std::shared_ptr<Camera> Ptr;
    float fx_, fy_, cx_, cy_, s_, depth_scale_;

    Camera();
    Camera(float fx, float fy, float cx, float cy, float s = 0, float depth_scale = 1):
    fx_(fx), fy_(fy), cx_(cx), cy_(cy), s_(s), depth_scale_(depth_scale)
    {}
    ~Camera();

    cv::Point2f pixel2cam_cv( const cv::Point2d& p, const cv::Mat& K );

    Vector3d world2cam(const SE3& T_c_w, const Vector3d& point_world);
    Vector2d cam2pixel(const Vector3d& point_camera);
    Vector3d pixel2cam(const Vector2d& point_image, double depth = 1);
    Vector3d cam2world(const SE3& T_c_w, const Vector3d& point_camera);

    Vector2d world2pixel(const SE3& T_c_w, const Vector3d& point_world);
    Vector3d pixel2world(const SE3& T_c_w, const Vector2d& point_image, double depth =1 );

};

}
```

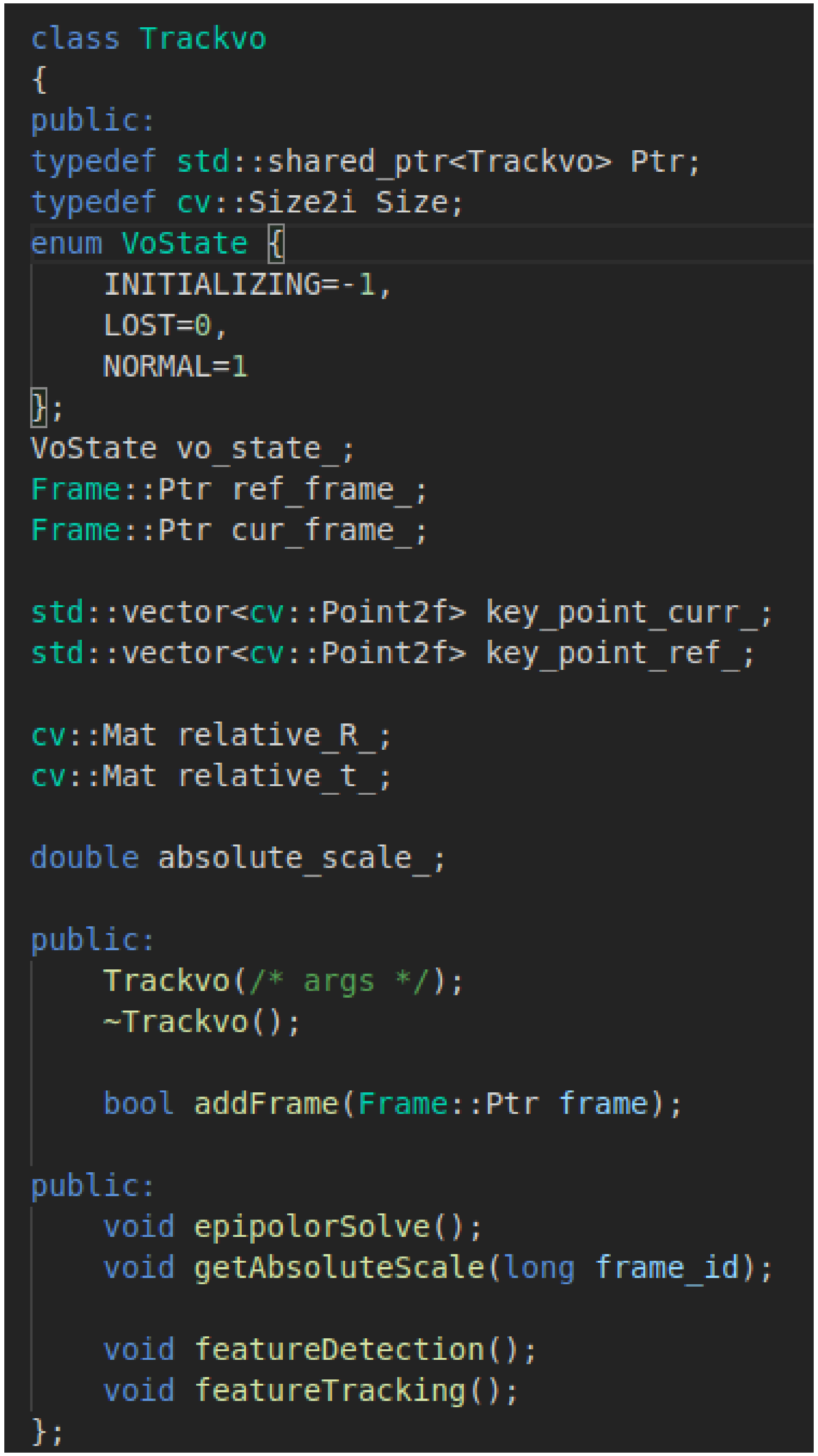

```
namespace slam
{

class MapPoint;
class Frame
{
private:
    /* data */
public:
    typedef std::shared_ptr<Frame> Ptr;
    long id_;
    cv::Mat rgb_, depth_;
    cv::Mat R_,t_;
    Camera::Ptr cam_;
    SE3 T_c_w_;
    cv::Mat Tcw;
    double time_stamp_;

public:
    Frame();
    Frame(long id, double time_stamp=0, SE3 T_c_w=SE3(),
     Camera::Ptr cam=nullptr, Mat color=Mat(), Mat depth=Mat());
    ~Frame();
    static Frame::Ptr createFrame();
    double findDepth(const cv::KeyPoint& kp);
    Vector3d getCameraCenter() const;
    bool isInFrame(const Vector3d& pt_world);
    void se3ToT34();
};

}
```



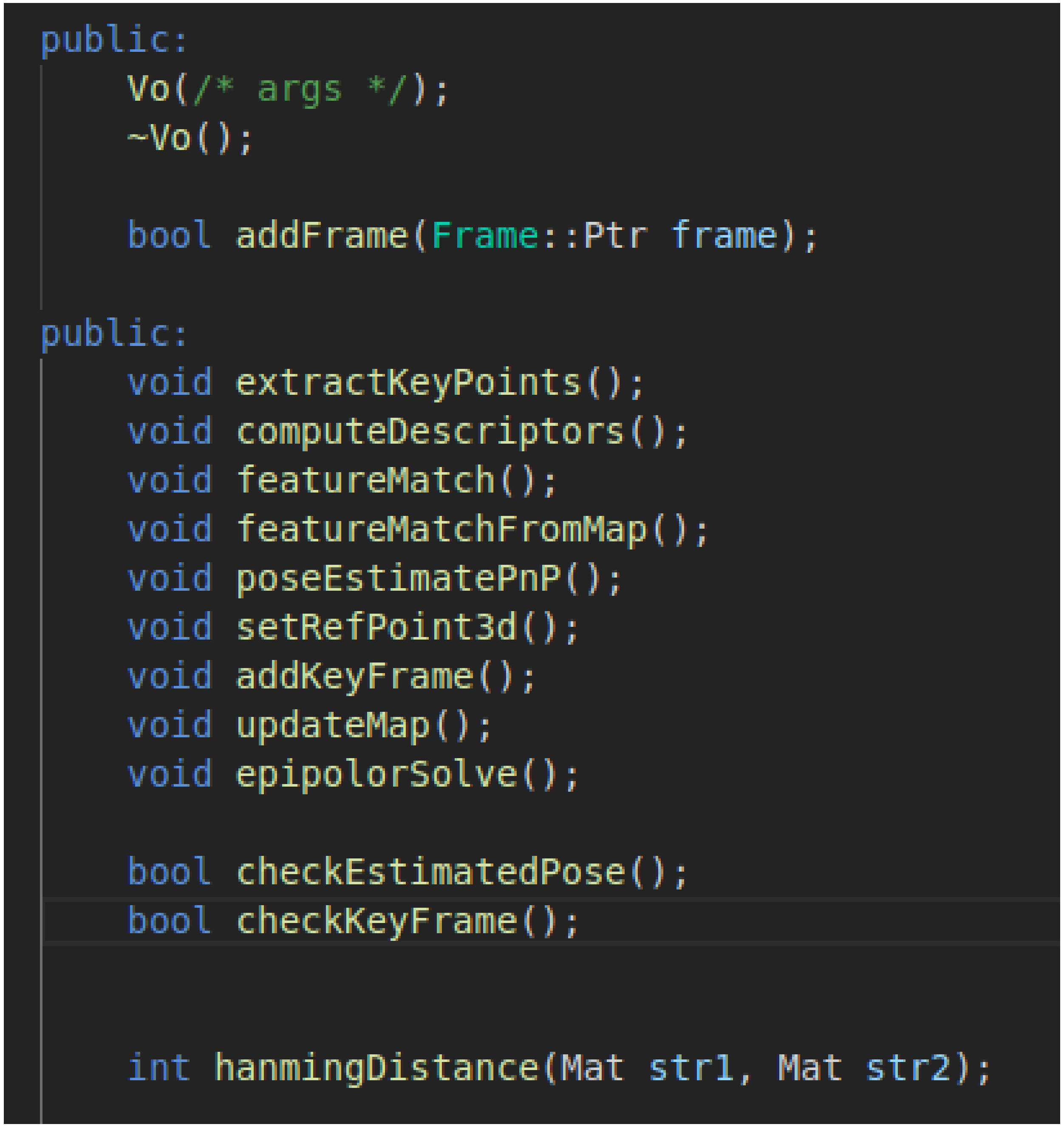
```cpp
namespace slam
{
class Mappoint
{
private:
    /* data */
public:
    typedef std::shared_ptr<Mappoint> Ptr;
    unsigned long id_;
    long _frame_id;
    Vector3d point_pos_;
    Vector3d view_direction_;
    int observed_times_;
    int correct_times_;
    Mat descriptor_;
public:
    Mappoint();
    Mappoint(long id, Vector3d point_pos, Vector3d view_direction);
    Mappoint(long id, long frame_id, Vector3d point_pos, Vector3d view_direction);
    inline cv::Point3f getPositionCV() const {
        return cv::Point3f( point_pos_(0,0), point_pos_(1,0), point_pos_(2,0) );
    }
    ~Mappoint();

    static Mappoint::Ptr createPoint();
    static Mappoint::Ptr createPoint( Vector3d point_poss);
    static Mappoint::Ptr createPoint( Vector3d point_poss, long frame_id);

};

} // myslam
```



```cpp
namespace slam
{

class Localmap
{
public:
    int _max_key_frames;
    std::vector<Frame::Ptr> key_frames_;
    std::vector<Mappoint::Ptr> map_points_;
public:
    typedef std::shared_ptr<Localmap> Ptr;
    Localmap() {}
    ~Localmap() {}
    Localmap(int max_key_frames)
    :_max_key_frames(max_key_frames)
    {

    }
    void addKeyFrame(Frame::Ptr frame);
    void addMapPoint(Mappoint::Ptr point);
};
} // namespace slam
```



```cpp
class Trackvo
{
public:
typedef std::shared_ptr<Trackvo> Ptr;
typedef cv::Size2i Size;
enum VoState {
    INITIALIZING=-1,
    LOST=0,
    NORMAL=1
};
VoState vo_state_;
Frame::Ptr ref_frame_;
Frame::Ptr cur_frame_;

std::vector<cv::Point2f> key_point_curr_;
std::vector<cv::Point2f> key_point_ref_;

cv::Mat relative_R_;
cv::Mat relative_t_;

double absolute_scale_;

public:
    Trackvo(/* args */);
    ~Trackvo();

    bool addFrame(Frame::Ptr frame);

public:
    void epipolorSolve();
    void getAbsoluteScale(long frame_id);

    void featureDetection();
    void featureTracking();
};
```



```
public:
    Vo(/* args */);
    ~Vo();

    bool addFrame(Frame::Ptr frame);

public:
    void extractKeyPoints();
    void computeDescriptors();
    void featureMatch();
    void featureMatchFromMap();
    void poseEstimatePnP();
    void setRefPoint3d();
    void addKeyFrame();
    void updateMap();
    void epipolorSolve();

    bool checkEstimatedPose();
    bool checkKeyFrame();


    int hanmingDistance(Mat str1, Mat str2);
```