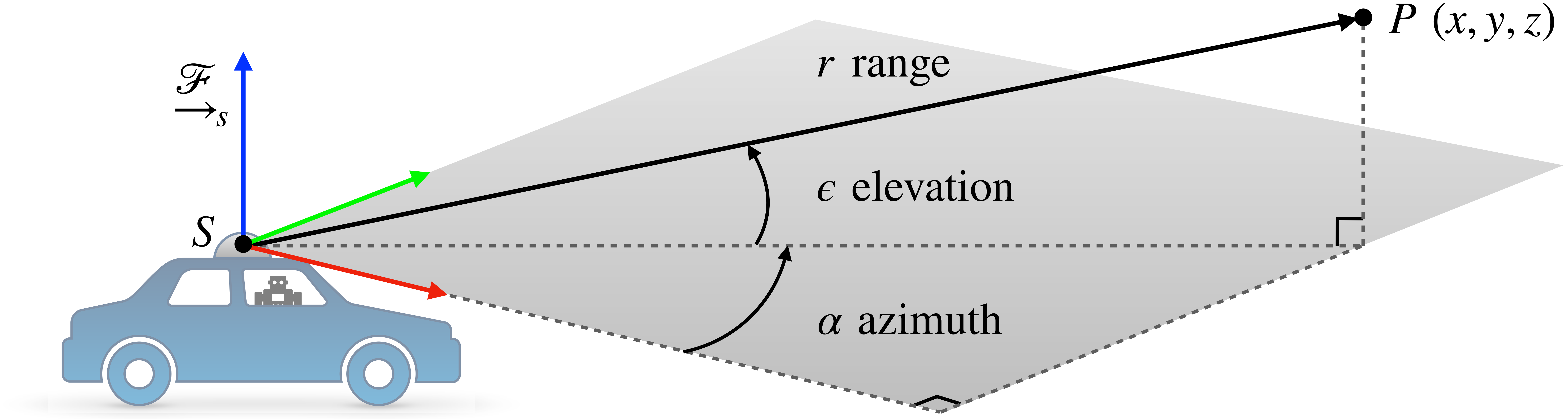
MODULE 4 LESSON 2

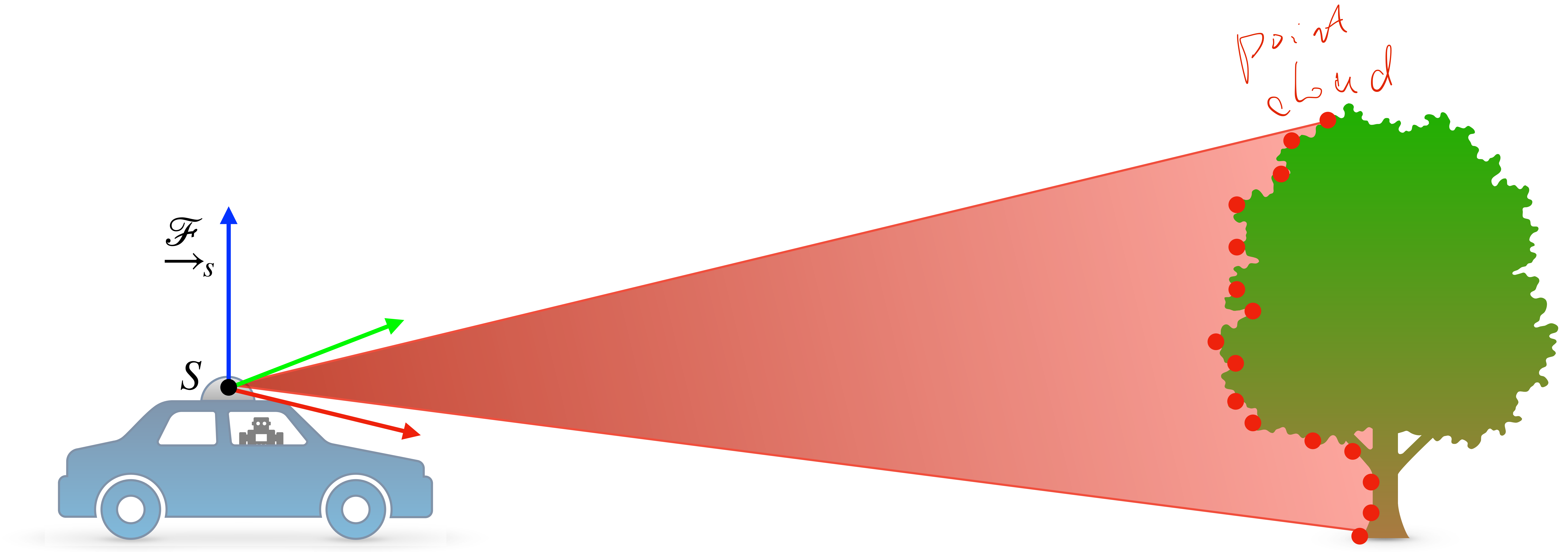
LIDAR SENSOR MODELS AND POINT CLOUDS

LIDAR Point Clouds



$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{h}^{-1}(r, \alpha, \epsilon) = \begin{bmatrix} r\cos\alpha\cos\epsilon \\ r\sin\alpha\cos\epsilon \\ r\sin\epsilon \end{bmatrix}$$

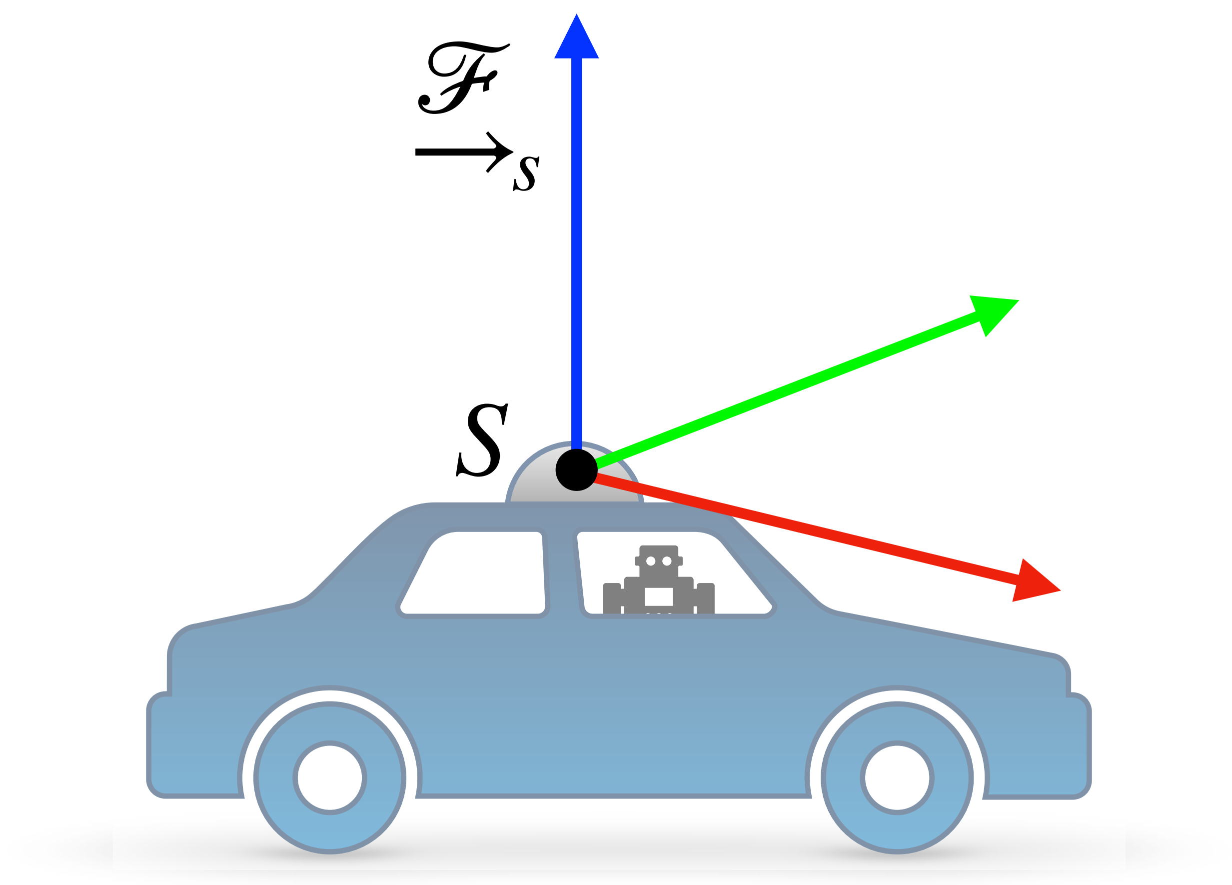
LIDAR Point Clouds

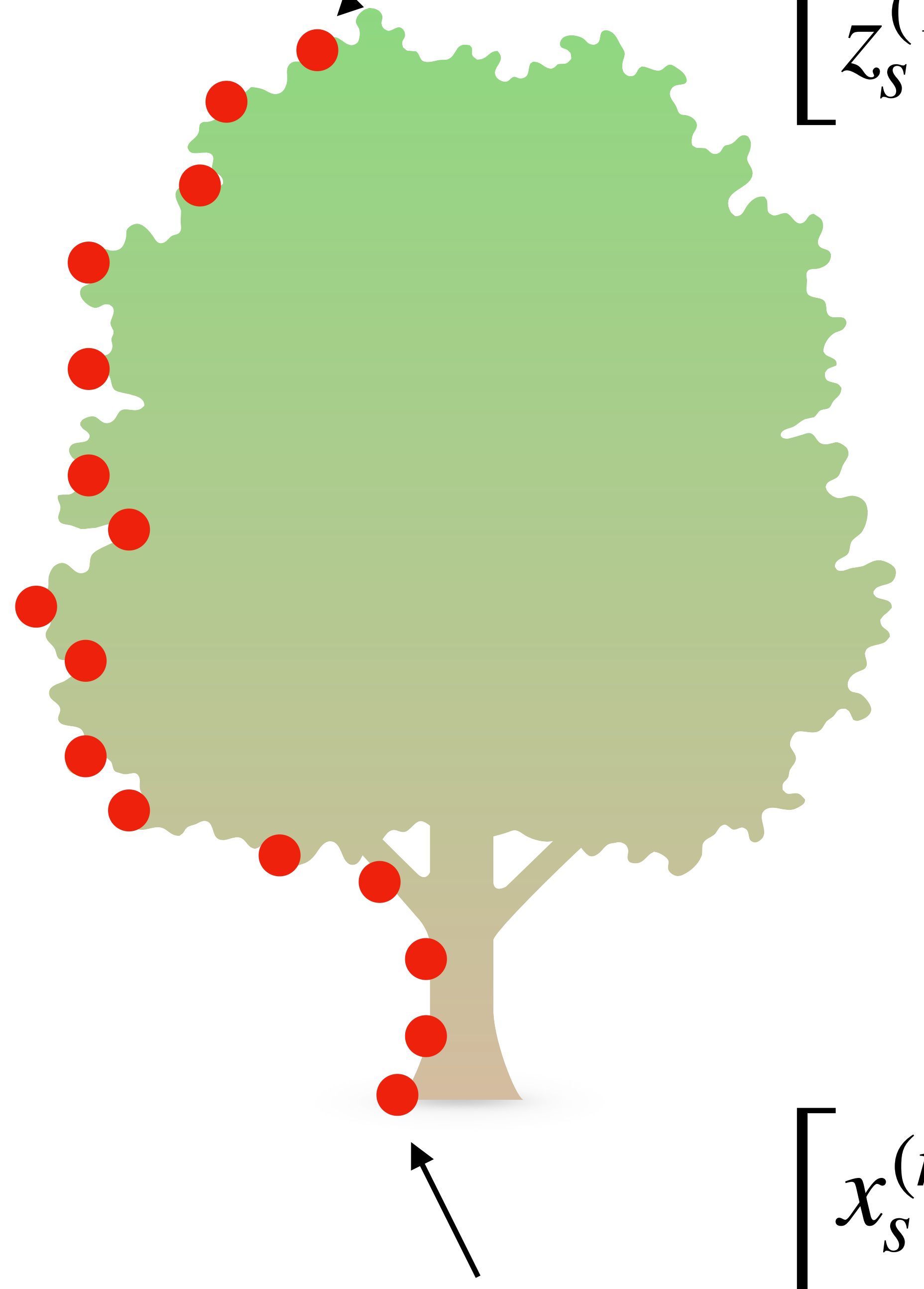


LIDAR Point Clouds | Data Structures

$$\mathbf{P}_s = \begin{bmatrix}\mathbf{p}_s^{(1)} & \mathbf{p}_s^{(2)} & \cdots & \mathbf{p}_s^{(n)}\end{bmatrix} = \begin{bmatrix} x_s^{(1)} & x_s^{(2)} & \cdots & x_s^{(n)} \\ y_s^{(1)} & y_s^{(2)} & \cdots & y_s^{(n)} \\ z_s^{(1)} & z_s^{(2)} & \cdots & z_s^{(n)} \end{bmatrix}$$

One point



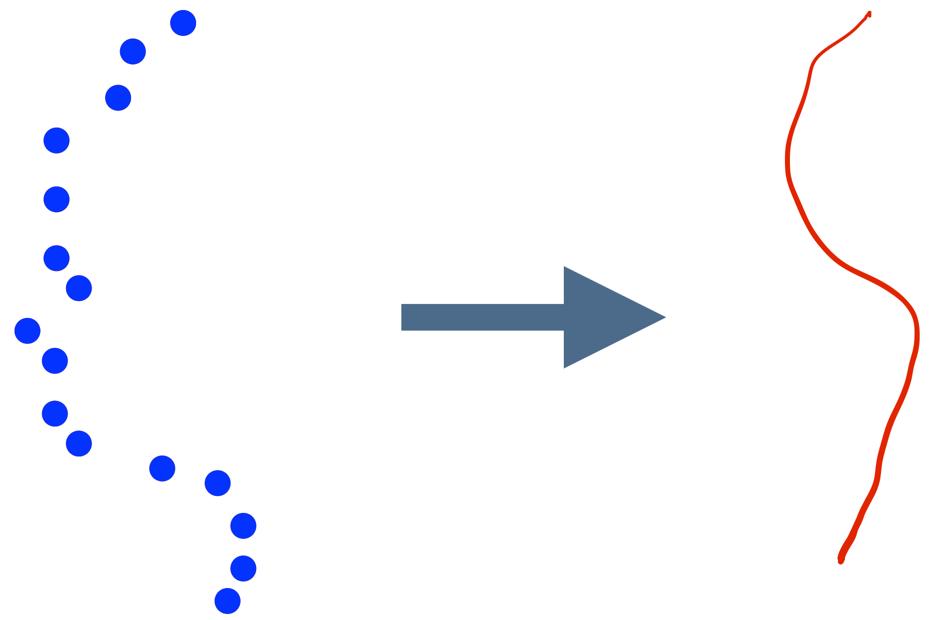


$$\mathbf{p}_s^{(1)} = \begin{bmatrix} x_s^{(1)} \\ y_s^{(1)} \\ z_s^{(1)} \end{bmatrix}$$

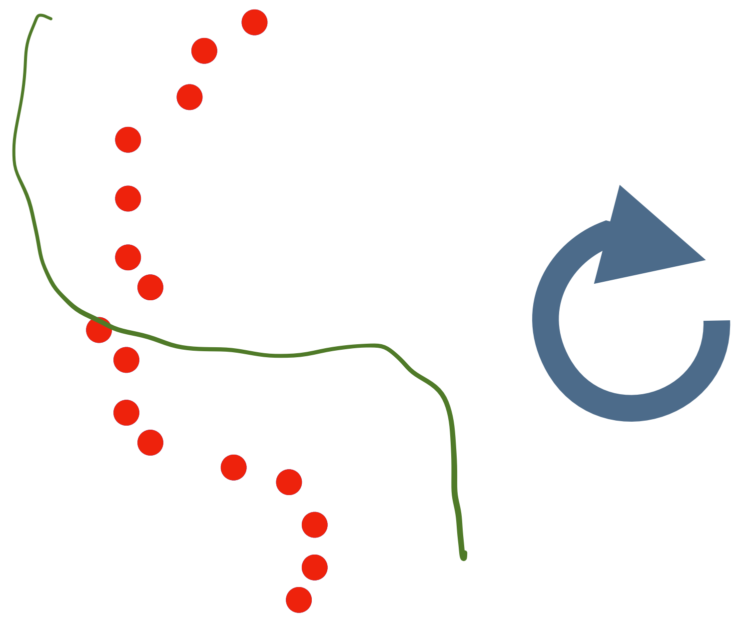
$$\mathbf{p}_s^{(n)} = \begin{bmatrix} x_s^{(n)} \\ y_s^{(n)} \\ z_s^{(n)} \end{bmatrix}$$

Operations on Point Clouds

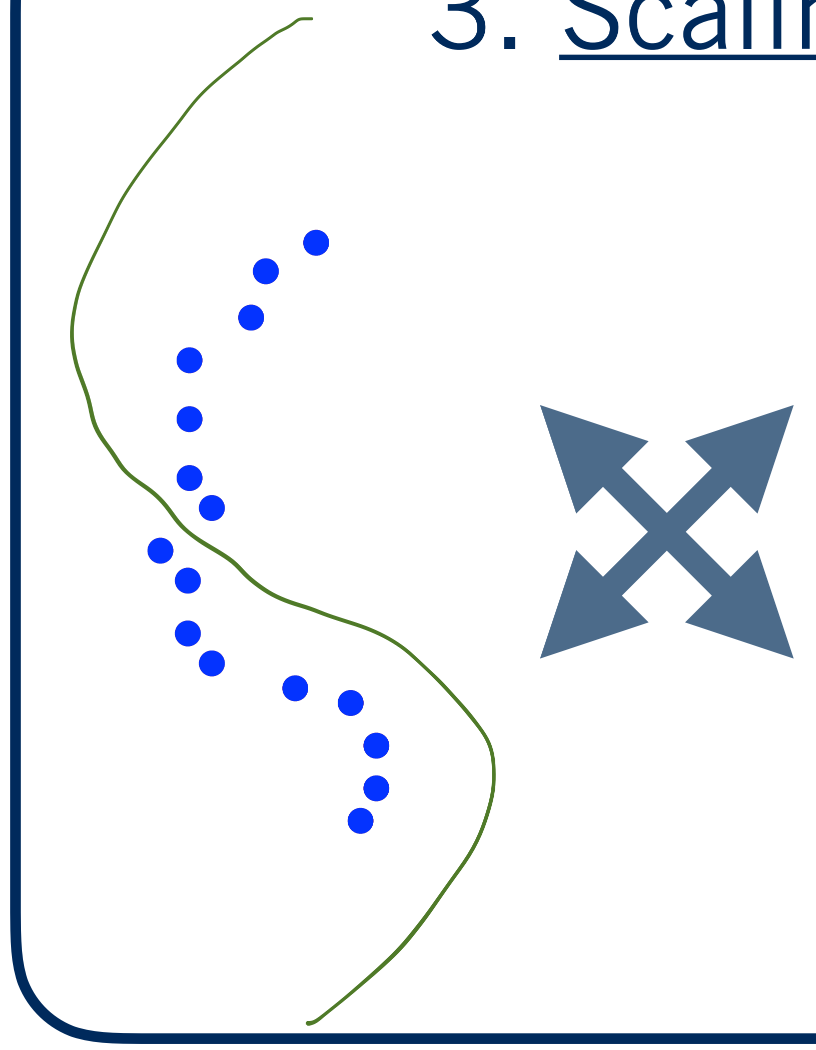
1. Translation



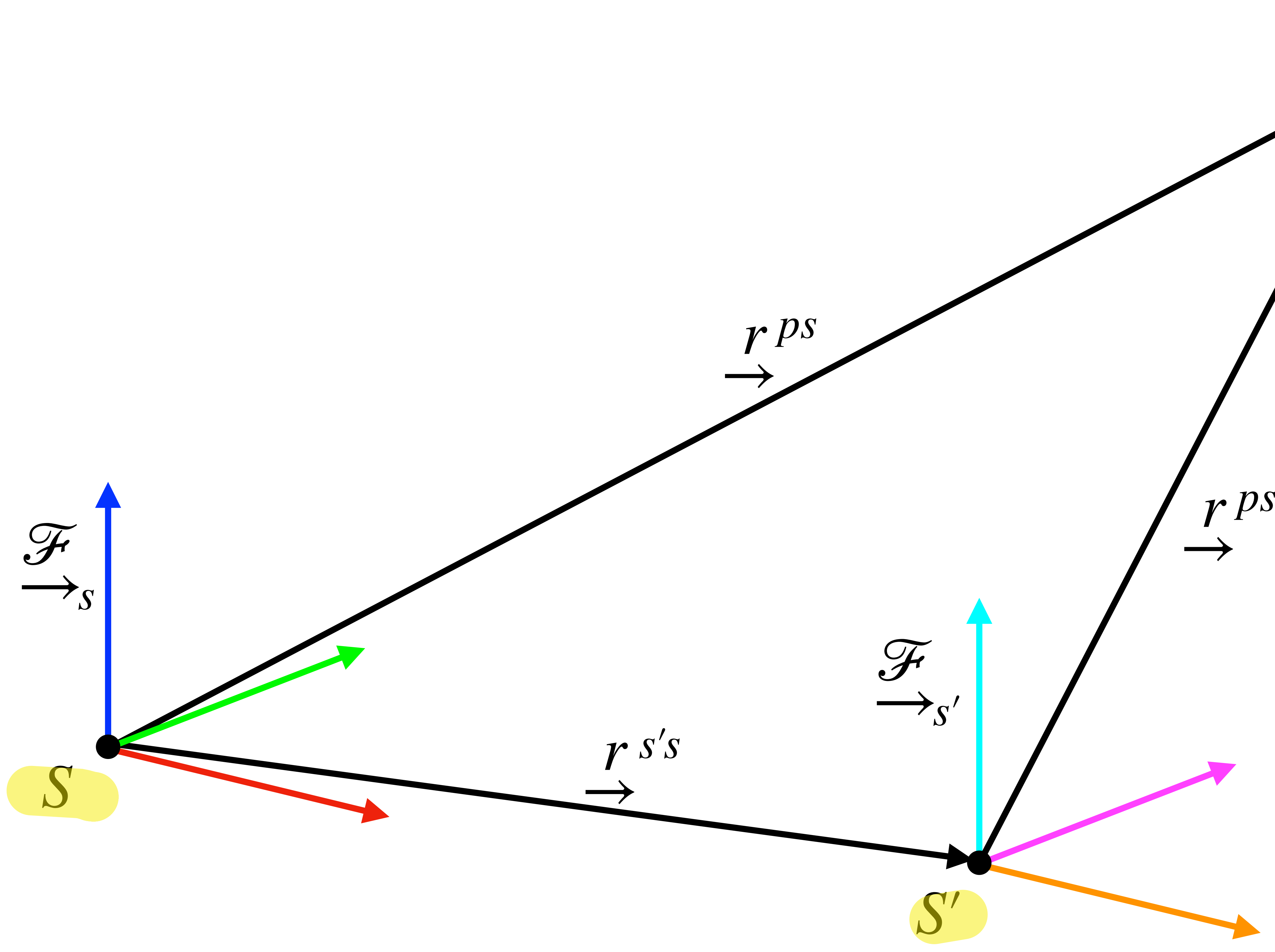
2. Rotation



3. Scaling



Operations on Point Clouds | Translation



From vector addition

$$\underset{\rightarrow}{r}^{ps'} = \underset{\rightarrow}{r}^{ps} - \underset{\rightarrow}{r}^{s's}$$

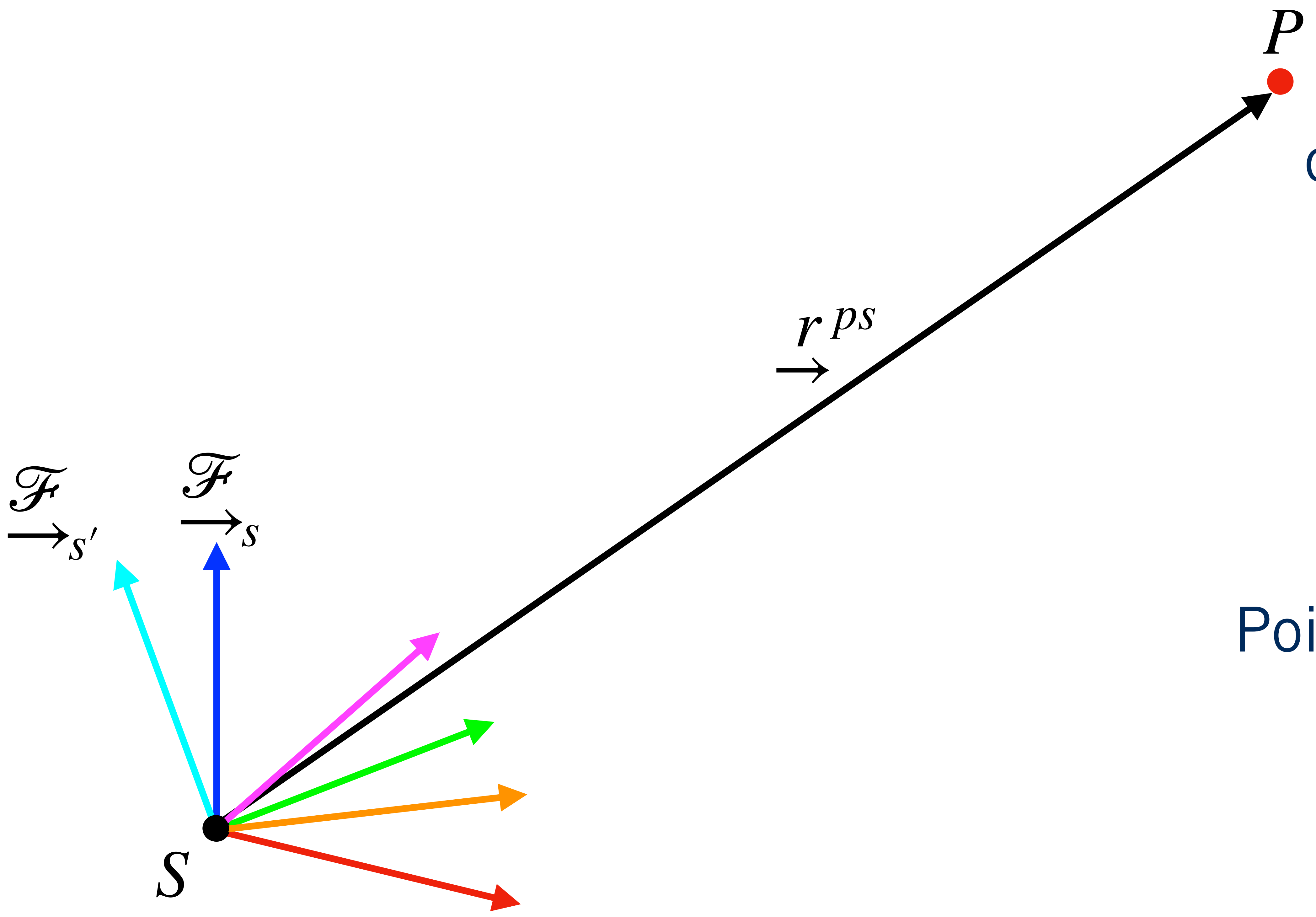
Point coordinates in the translated frame

$$\mathbf{p}_{s'}^{(j)} = \mathbf{p}_s^{(j)} - \mathbf{r}_s^{s's} \quad \text{each point}$$

$$\mathbf{P}_{s'} = \mathbf{P}_s - \mathbf{R}_s^{s's} \quad \text{whole cloud}$$

$$\mathbf{R}_s^{s's} = \begin{bmatrix} \mathbf{r}_s^{s's} & \mathbf{r}_s^{s's} & \cdots & \mathbf{r}_s^{s's} \end{bmatrix}$$

Operations on Point Clouds | Rotation



The *rotation matrix* takes coordinates from frame *s* to frame *s'*

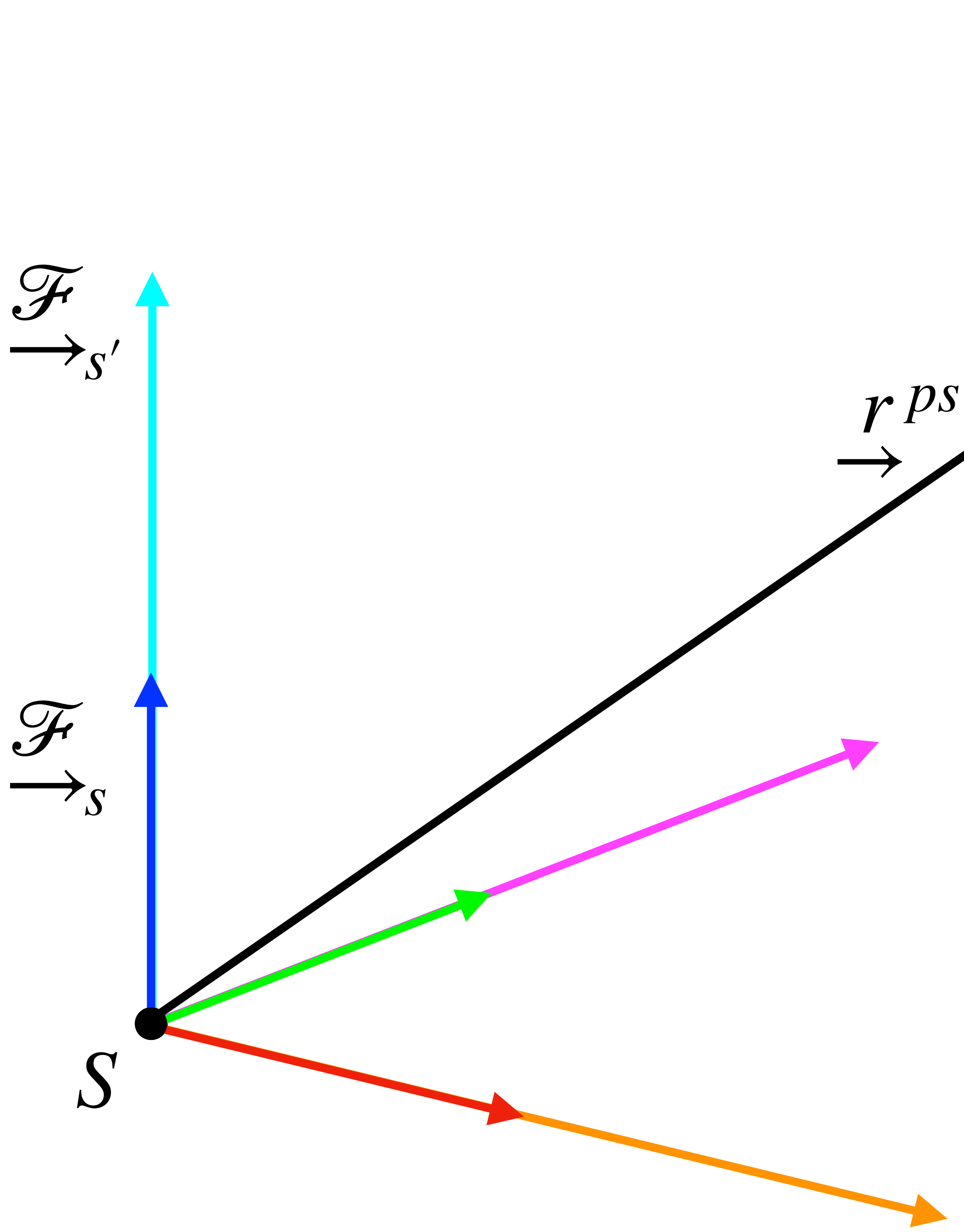
$$\mathbf{r}_{s'} = \mathbf{C}_{s's}\,\mathbf{r}_s$$

Point coordinates in the rotated frame

$$\mathbf{p}_{s'}^{(j)} = \mathbf{C}_{s's}\,\mathbf{p}_s^{(j)}$$ each point

$$\mathbf{P}_{s'} = \mathbf{C}_{s's}\,\mathbf{P}_s$$ whole cloud

Operations on Point Clouds | Scaling



The *scaling matrix* is composed of scaling factors for each basis vector of frame *s*

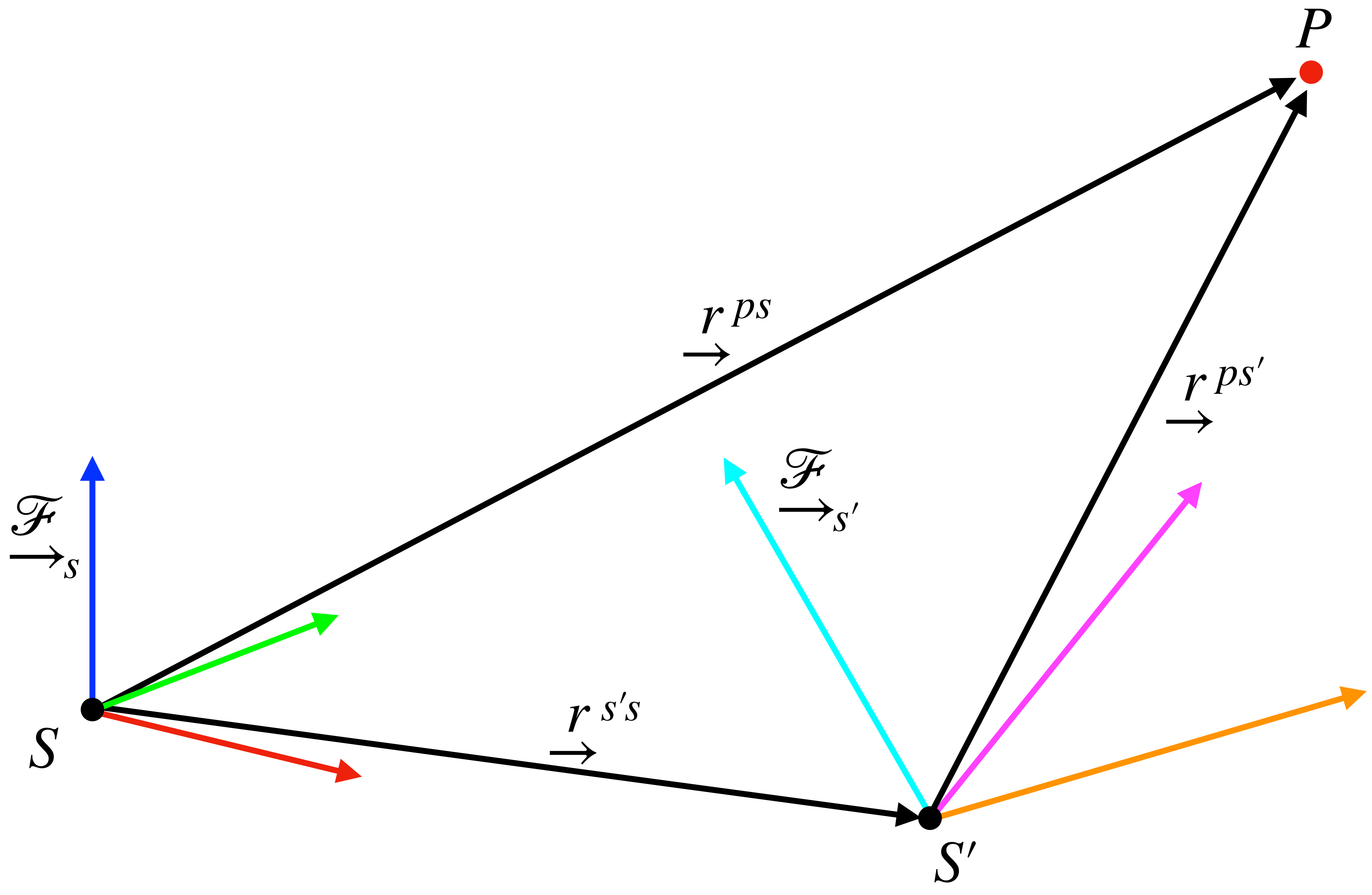
$$\mathbf{r}_{s'} = \underbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}}_{\mathbf{S}_{s's}} \mathbf{r}_s$$

Point coordinates in the scaled frame

$$\mathbf{p}_{s'}^{(j)} = \mathbf{S}_{s's}\, \mathbf{p}_s^{(j)} \qquad \text{each point}$$

$$\mathbf{P}_{s'} = \mathbf{S}_{s's}\, \mathbf{P}_s \qquad \text{whole cloud}$$

Operations on Point Clouds | Putting Them All Together



Point coordinates in the transformed frame

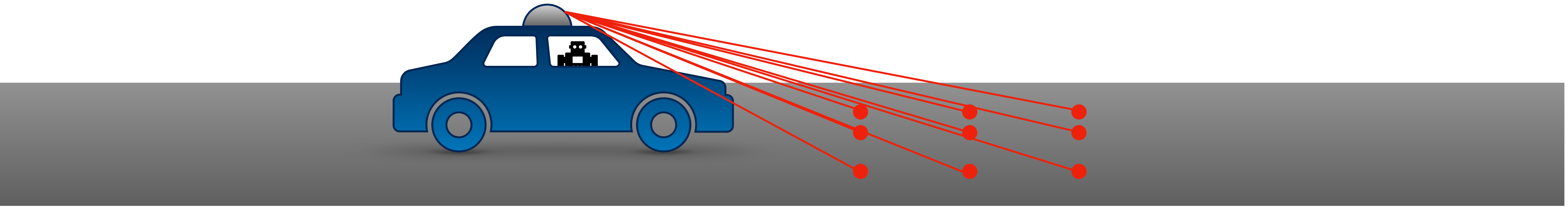
$$\mathbf{p}_{s'}^{(j)} = \mathbf{S}_{s's}\mathbf{C}_{s's}\left(\mathbf{p}_s^{(j)} - \mathbf{r}_s^{s's}\right)$$ each point

3. Scale 2. Rotate 1. Translate

$$\mathbf{P}_{s'} = \mathbf{S}_{s's}\mathbf{C}_{s's}\left(\mathbf{P}_s - \mathbf{R}_s^{s's}\right)$$ whole cloud

Finding the Road with 3D Plane Fitting

Where is the road now? Where is it going to be?

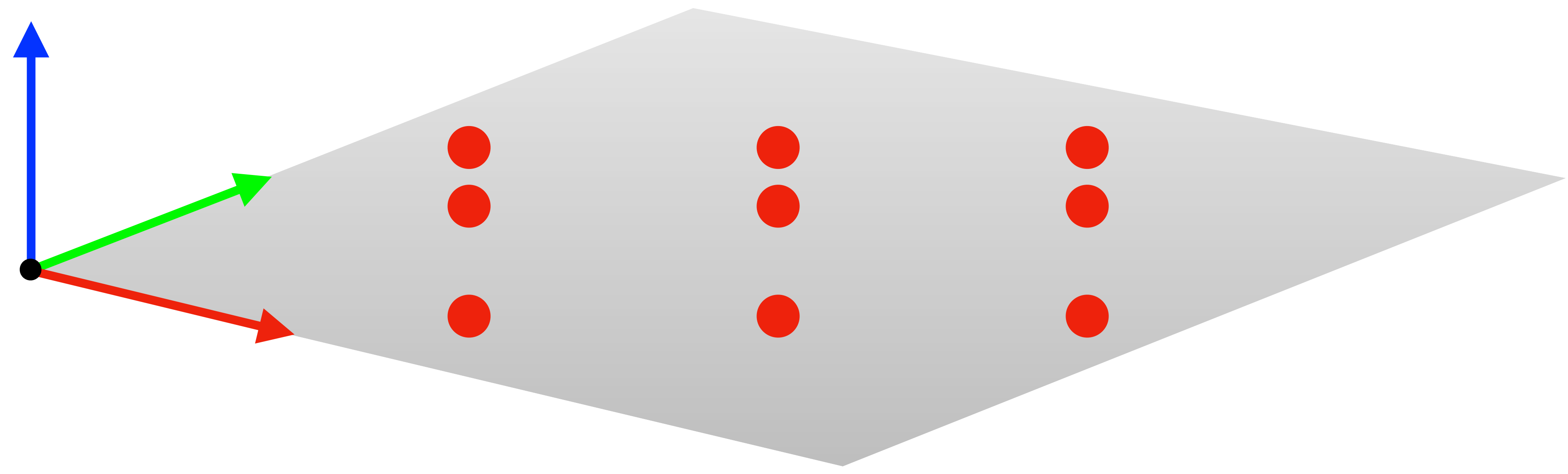


Finding the Road with 3D Plane Fitting

We have measurements of (x, y, z) and we want to determine the parameters (a, b, c) — *use least-squares*!

Equation of a plane in 3D:

$$z = a + bx + cy$$



Measurement error:

$$e_j = \hat{z}_j - z_j$$

observed value

$$= \left(\hat{a} + \hat{b}x_j + \hat{c}y_j\right) - z_j \qquad j = 1 \ldots n$$

predicted

Not account for lidar sensor noise

Finding the Road with 3D Plane Fitting

We can stack all of the measurement errors into matrix form

$$\underbrace{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}}_{\mathbf{e}} = \underbrace{\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & y_n \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{\mathbf{x}} - \underbrace{\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}}_{\mathbf{b}}$$

And minimize the squared-error criterion to get the least-squares solution for the parameters

$$\begin{aligned} \hat{\mathbf{x}} &= \operatorname{argmin}_{\mathbf{x}} \mathscr{L}_{\mathrm{LS}}(\mathbf{x}) \\ \mathscr{L}_{\mathrm{LS}}(\mathbf{x}) &= \mathbf{e}^T\mathbf{e} \\ &= \left(\mathbf{A}\mathbf{x} - \mathbf{b}\right)^T \left(\mathbf{A}\mathbf{x} - \mathbf{b}\right) \\ &= \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{A}^T\mathbf{b} - \mathbf{b}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{b} \end{aligned}$$

Finding the Road with 3D Plane Fitting

$$\boxed{\begin{aligned}\hat{\mathbf{x}} &= \operatorname{argmin}_{\mathbf{x}} \mathscr{L}_{\mathrm{LS}}(\mathbf{x}) \\ \mathscr{L}_{\mathrm{LS}}(\mathbf{x}) &= \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{A}^T\mathbf{b} - \mathbf{b}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{b}\end{aligned}}$$

Taking the partial derivative with respect to **x** and setting to zero for an optimum gives us the familiar *normal equations*

$$\left.\frac{\partial \mathscr{L}_{\mathrm{LS}}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}} = 2\mathbf{A}^T\mathbf{A}\hat{\mathbf{x}} - 2\mathbf{A}^T\mathbf{b} = \mathbf{0} \quad \longrightarrow \quad \mathbf{A}^T\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^T\mathbf{b}$$

We can solve for **x** using an efficient numerical solver or by using the *pseudo-inverse*

$$\hat{\mathbf{x}} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}$$

The Point Cloud Library (PCL)

pcl

- Open-source Point Cloud Library (PCL) has many useful functions for doing basic and advanced operations on point clouds in C++
- Widely used in industry
- Unofficial Python bindings exist

Summary | Point Clouds

- The Cartesian coordinates of all the measurements from a LIDAR scan are stored in a *point cloud*
- Point clouds can be translated, rotated, or scaled
- We can use point clouds for useful self-driving tasks, like fitting a 3D plane to find the road surface
- The Point Cloud Library (PCL) implements many useful tools for working with points clouds in C++