GP-Tag Technical Specification

S. E. Sundén Byléhn

January 5, 2025

1 Introduction

The GP-Tag is a fiducial marker designed for accurate pose estimation and geospatial data encoding. With a base unit of U=1, the tag has total dimensions of exactly 36×36 units, composed of a central 21×21 grid surrounded by concentric rings and corner spikes.

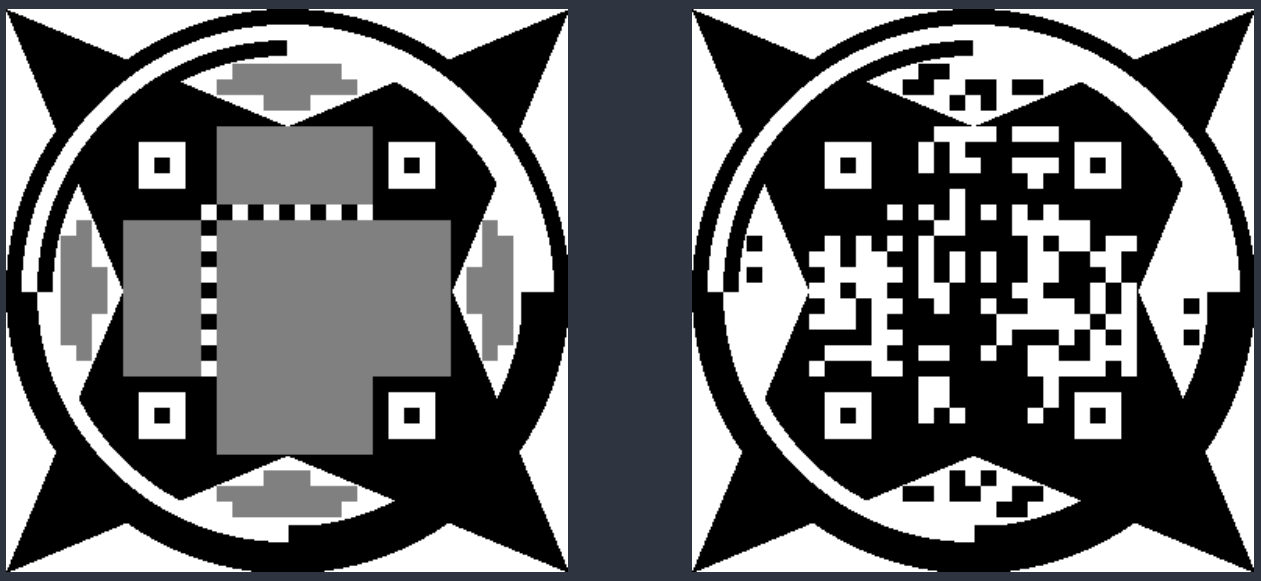

Figure 1: Left: Basic GP-Tag structure emphasizing the geometric patterns with the data cells grayed out. Right: GP-Tag with encoded data cells. (Shown with U=10)

For practical applications, these dimensions are scaled by U, with a recommended minimum of U=10 pixels. The example figures above use U=10, resulting in a tag size of 360×360 pixels.

2 Coordinate Systems

Warning: Implementers must stay aware of coordinate conventions and recognize potential discrepancies in Y-axis orientation, rotation direction, indexing base, and notation styles. Using the GP-Tag requires attention to these conventions, with particular regard to the Robotics Convention (RC) for Y-Up and the Computer Vision Convention (CVC) for Y-Down. Each library's specifics—whether Y-Up or Y-Down, clockwise or counterclockwise rotation, 0-based or 1-based indexing, and Row-Column or X-Y notation—must be confirmed to ensure accurate data handling.

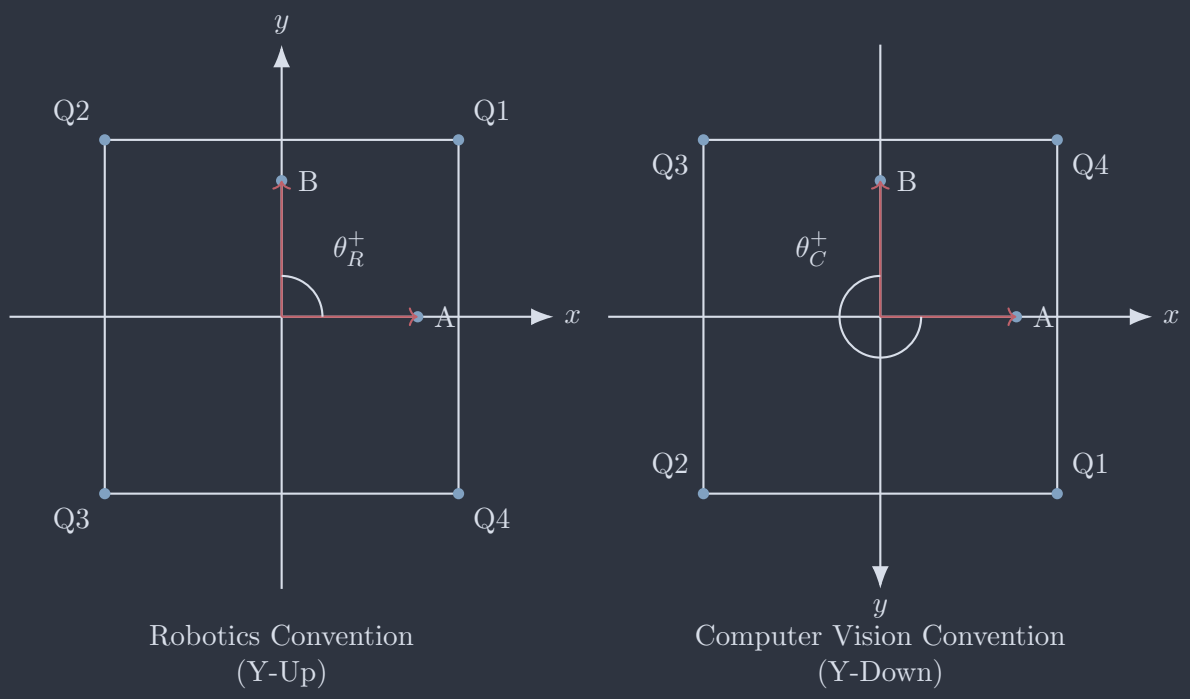

Figure 2: GP-Tag Coordinate Systems

The fundamental differences between the two conventions are:

- **Origin:** Both systems place the origin at the tag center
- **Y-axis Direction:** RC uses upward positive Y, CVC uses downward positive Y
- **X-axis Direction:** Both systems use rightward positive X
- **Rotation Convention:** RC uses counter-clockwise from +X, CVC uses clockwise from +X
- **Z-axis:** RC points outward from tag surface, CVC points inward (right-hand rule)

2.1 Grid Indexing Relationship

The relationship between coordinate systems affects how grid rows are indexed. In grid-based notation, unlike coordinate-based systems that center the origin, indexing starts from one of the grid corners. Additionally, indexing may start at either 0 or 1, depending on the convention used:

- **RC (Y-Up):** Row indices increase from bottom to top
  - Zero-based: Row 0 at bottom, Row 20 at top
  - One-based: Row 1 at bottom, Row 21 at top
- **CVC (Y-Down):** Row indices increase from top to bottom
  - Zero-based: Row 0 at top, Row 20 at bottom
  - One-based: Row 1 at top, Row 21 at bottom
- **Column Indexing:** Identical in both systems

- Zero-based: Column 0 to 20, left to right
- One-based: Column 1 to 21, left to right

2.2 Library-Specific Conventions

Certain libraries have established conventions that differ in Y-axis orientation, rotation direction, and coordinate indexing. Below are the default conventions for some commonly used libraries:

Table 1: Library Coordinate and Indexing Conventions

Library	Y-Axis Orientation	Rotation Direction	Indexing Base
OpenCV	Y-Down	Clockwise	0-based
PIL	Y-Down	Clockwise	0-based
NumPy	Y-Up	Counterclockwise	0-based
ROS2	Y-Up	Counterclockwise	0-based

Verify library conventions to ensure consistent GP-Tag coordinate handling.

3 Tag Definition in RC

The RC system uses Y-Up convention with:

- Row indices increase from bottom to top (row 1 at bottom, row 21 at top)
- Column indices increase from left to right (column 1 to 21)

3.1 Geometrical Spatial Markers

3.1.1 Corner Definition

In the RC system, with radius R_i (inner radius) from the center to the grid boundary:

$$R_i = \frac{\sqrt{(21U)^2 + (21U)^2}}{2} = \frac{21U\sqrt{2}}{2} \approx 14.85U \tag{1}$$

where $21U$ represents the full grid size (21 units $\times$ scale factor U). To ensure no intersection with grid cells, we round up this radius to:

$$R_i = 15U \tag{2}$$

This defines the inner circle of the inner annulus and serves as the basic reference for all other tag dimensions.

Corner positions of the grid (not the full tag) are defined as:

$$\begin{aligned} \text{Q1 (Top Right):} \quad & \mathbf{p}_1^R = (+R_i, +R_i) \\ \text{Q2 (Top Left):} \quad & \mathbf{p}_2^R = (-R_i, +R_i) \\ \text{Q3 (Bottom Left):} \quad & \mathbf{p}_3^R = (-R_i, -R_i) \\ \text{Q4 (Bottom Right):} \quad & \mathbf{p}_4^R = (+R_i, -R_i) \end{aligned}$$

The outer radius R_o which defines the tip of the spikes is:

$$R_o = R_i + 3U \tag{3}$$

This means the total tag dimensions are $(2R_o \times 2R_o)$ units.

3.1.2 Spike Definition

Each spike is a triangular extension from the tag's perimeter, defined by three points:

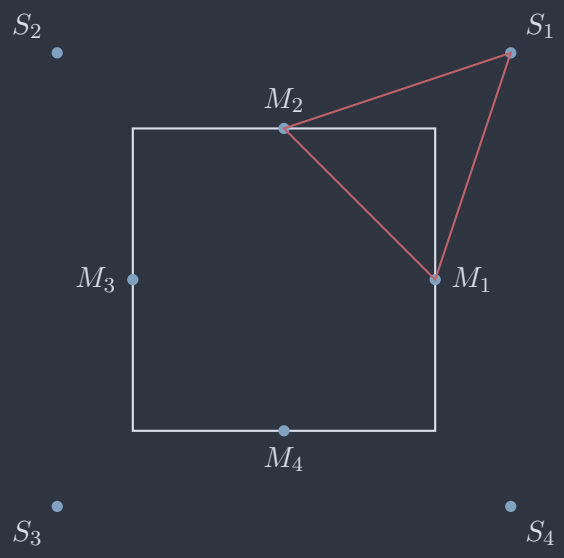

Figure 3: Spike Triangle Definition (showing Q1 spike)

For spike tips at outer radius R_o (where $R_o = d + 3U$):

$$\begin{aligned} \text{Spike Tips:} \quad & S_1(+R_o, +R_o),\ S_2(-R_o, +R_o),\ S_3(-R_o, -R_o),\ S_4(+R_o, -R_o) \\ \text{Midpoints:} \quad & M_1(+d, 0),\ M_2(0, +d),\ M_3(-d, 0),\ M_4(0, -d) \end{aligned}$$

Each spike triangle is defined by:

$$T_i = \{S_i, M_i, M_{(i \bmod 4)+1}\}, \quad i = 1, 2, 3, 4 \tag{4}$$

3.1.3 Annuli

The tag contains three concentric annular regions:

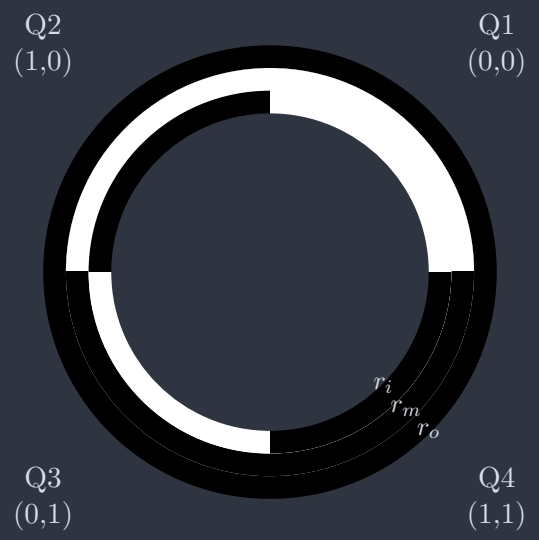

Figure 4: Annuli Pattern Structure

3.1.4 Inner and Middle Annuli

These two annuli encode orientation information through binary patterns in each quadrant:
Middle Ring Pattern (counter-clockwise from Q1):

Q1 (+x,+y):	0 (white)
Q2 (-x,+y):	0 (white)
Q3 (-x,-y):	1 (black)
Q4 (+x,-y):	1 (black)

Inner Ring Pattern (counter-clockwise from Q1):

Q1 (+x,+y):	0 (white)
Q2 (-x,+y):	1 (black)
Q3 (-x,-y):	0 (white)
Q4 (+x,-y):	1 (black)

3.1.5 Outer Annulus

The outer annulus is a solid black ring that serves as:

- Primary detection feature
- Initial circle detection target
- Boundary definition for the tag

3.2 Grid

3.2.1 Corner Markers

Each corner contains a finder pattern:

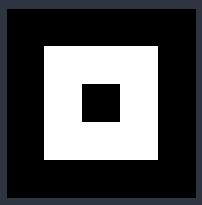

Figure 5: Finder Pattern Structure

3.2.2 Timing

Timing patterns consist of alternating black and white modules:

- Horizontal timing pattern: Located on row 16 (zero-based) / row 17 (one-based)
- Vertical timing pattern: Located on column 6 (zero-based) / column 7 (one-based)
- Both patterns span positions 6 through 14 (zero-based) / 7 through 15 (one-based)
- Used for grid alignment and perspective correction

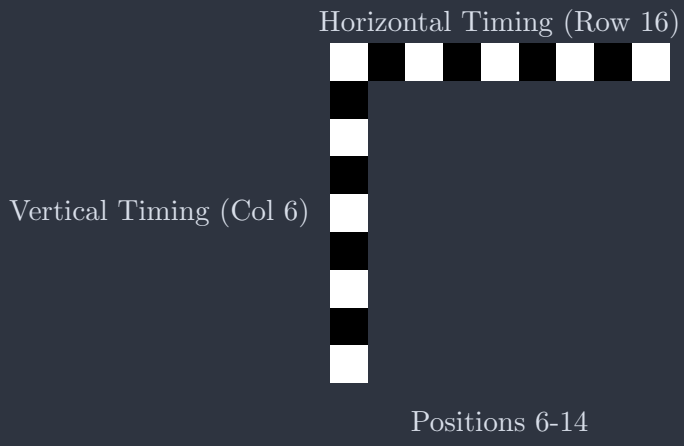

Figure 6: Timing Pattern Structure (zero-based, RC Y-up)

3.2.3 Separator Pattern

The separator pattern consists of two solid black lines that form an "L" shape:

- A horizontal black line along row 16
- A vertical black line along column 16

These provide high contrast edges to help identify and isolate the finder patterns.

3.2.4 Data Encoding

The GP-Tag encodes data in two distinct areas: the main grid (300 cells) and a reserved area (76 cells). These regions utilize Reed-Solomon error correction with a 50% overhead on byte-aligned data.

The tag's 21×21 grid includes corners, timing patterns, and separator markers, leaving 300 cells for main grid data encoding. Additionally, the tag contains four 19-cell regions within the annuli that are not occluded by other elements. We refer to these regions collectively as the reserved area, which currently encodes the Version ID and Tag ID. We group these four regions into two pairs of two and duplicate the data within each pair to ensure redundancy.

Table 2: Position and Orientation Data in Main Grid (300 cells)

Component	**Bits**	**Details**
Latitude	35	Range: $-90°$ to $+90°$, Resolution: $\approx 2.38 \times 10^{-9}$ degrees
Longitude	36	Range: $-180°$ to $+180°$, Resolution: $\approx 4.77 \times 10^{-9}$ degrees
Altitude	25	Range: -10000 m to $+10000$ m, Resolution: ≈ 0.6 mm
Quaternion	64 (4×16)	Range per component: -1 to $+1$, Resolution: $\approx 3.05 \times 10^{-5}$
Scale	16	Range: $[0.0036, 3.6]$ cells/mm, tag sizes from 10m to 1cm
Accuracy Indicator	2	4 discrete levels (0-3)

Table 3: Reserved Area Data (76 cells)

Component	**Bits**	**Details**
Tag ID	12	Range: 0 to 4095 unique IDs
Version ID	4	Range: 0 to 15 versions
Total	16	
Duplication total	32	

Table 4: Error Correction and Storage Requirements

Area	**Raw**	**Aligned**	**Err. Corr.**	**Total**	**Rem. Bits**
Main Grid	178 bits	184 bits	96 bits	280 bits	20 bits
Reserved Area	32 bits	32 bits	16 bits	48 bits	28 bits

Encoding Formulas

$$\text{Latitude Value} = \left\lfloor (\text{lat} + 90) \times \frac{2^{35} - 1}{180} \right\rfloor \tag{5}$$

$$\text{Longitude Value} = \left\lfloor (\text{lon} + 180) \times \frac{2^{36} - 1}{360} \right\rfloor \tag{6}$$

$$\text{Altitude Value} = \left\lfloor (\text{alt} + 10000) \times \frac{2^{25} - 1}{20000} \right\rfloor \tag{7}$$

$$\text{Quaternion Value}_i = \left\lfloor (q_i + 1) \times \frac{2^{16} - 1}{2} \right\rfloor \tag{8}$$

$$\text{Scale Value} = \left\lfloor \text{scale} \times \frac{2^{16} - 1}{3.6} \right\rfloor \tag{9}$$

3.2.5 Data Filling Order

Data cells in the main grid and reserved areas are filled as follows:

- **Main Grid (300 cells):**
  - Filling starts from the **top row, leftmost column** and proceeds **left to right** across each row.
  - At the end of each row, filling continues with the row directly below, following the **left-to-right** direction.
  - This order continues **row by row from the top to the bottom**, excluding reserved cells (corner markers, timing patterns, and separators).
- **Reserved Area (76 cells), PIL Y-Up (X, Y) Coordinates:**
  - Reserved cells are arranged in **Bottom/Top** and **Left/Right** mirrored pairs for redundancy.
  - The specific coordinates for each mirrored pair are as follows:

Table 5: Reserved Area Bottom/Top Pairs (Y-Up, X and Y Coordinates).
NOTE: Coordinates are in zero index convention.

Bottom Coordinate	Top Coordinate
(15, 4)	(21, 32)
(16, 4)	(20, 32)
(17, 4)	(19, 32)
(18, 4)	(18, 32)
(19, 4)	(17, 32)
(20, 4)	(16, 32)
(21, 4)	(15, 32)
(14, 5)	(22, 31)
(15, 5)	(21, 31)
(16, 5)	(20, 31)
(17, 5)	(19, 31)
(18, 5)	(18, 31)
(19, 5)	(17, 31)
(20, 5)	(16, 31)
(21, 5)	(15, 31)
(22, 5)	(14, 31)
(17, 6)	(19, 30)
(18, 6)	(18, 30)
(19, 6)	(17, 30)

Table 6: Reserved Area Left/Right Pairs (Y-Up, X and Y Coordinates).
NOTE: Coordinates are in zero index convention.

Left Coordinate	**Right Coordinate**
(4, 21)	(32, 15)
(4, 20)	(32, 16)
(4, 19)	(32, 17)
(4, 18)	(32, 18)
(4, 17)	(32, 19)
(4, 16)	(32, 20)
(4, 15)	(32, 21)
(5, 22)	(31, 14)
(5, 21)	(31, 15)
(5, 20)	(31, 16)
(5, 19)	(31, 17)
(5, 18)	(31, 18)
(5, 17)	(31, 19)
(5, 16)	(31, 20)
(5, 15)	(31, 21)
(5, 14)	(31, 22)
(6, 19)	(30, 17)
(6, 18)	(30, 18)
(6, 17)	(30, 19)

3.2.6 Data Filling Visualization

To illustrate the data filling order within the GP-Tag grid, Figure 7 shows the tag filled with a color gradient, progressing from blue to red. This gradient highlights the sequence in which cells are filled. For the main grid, beginning in the top-left corner and moving left-to-right across each row. Once a row is filled, the filling proceeds to the next row below, following the same direction.

In the reserved areas, pairs of cells (mirrored top-bottom and left-right) are filled in the specified order, ensuring redundancy. These mirrored pairs are visible as matching colors on opposite sides of the tag.

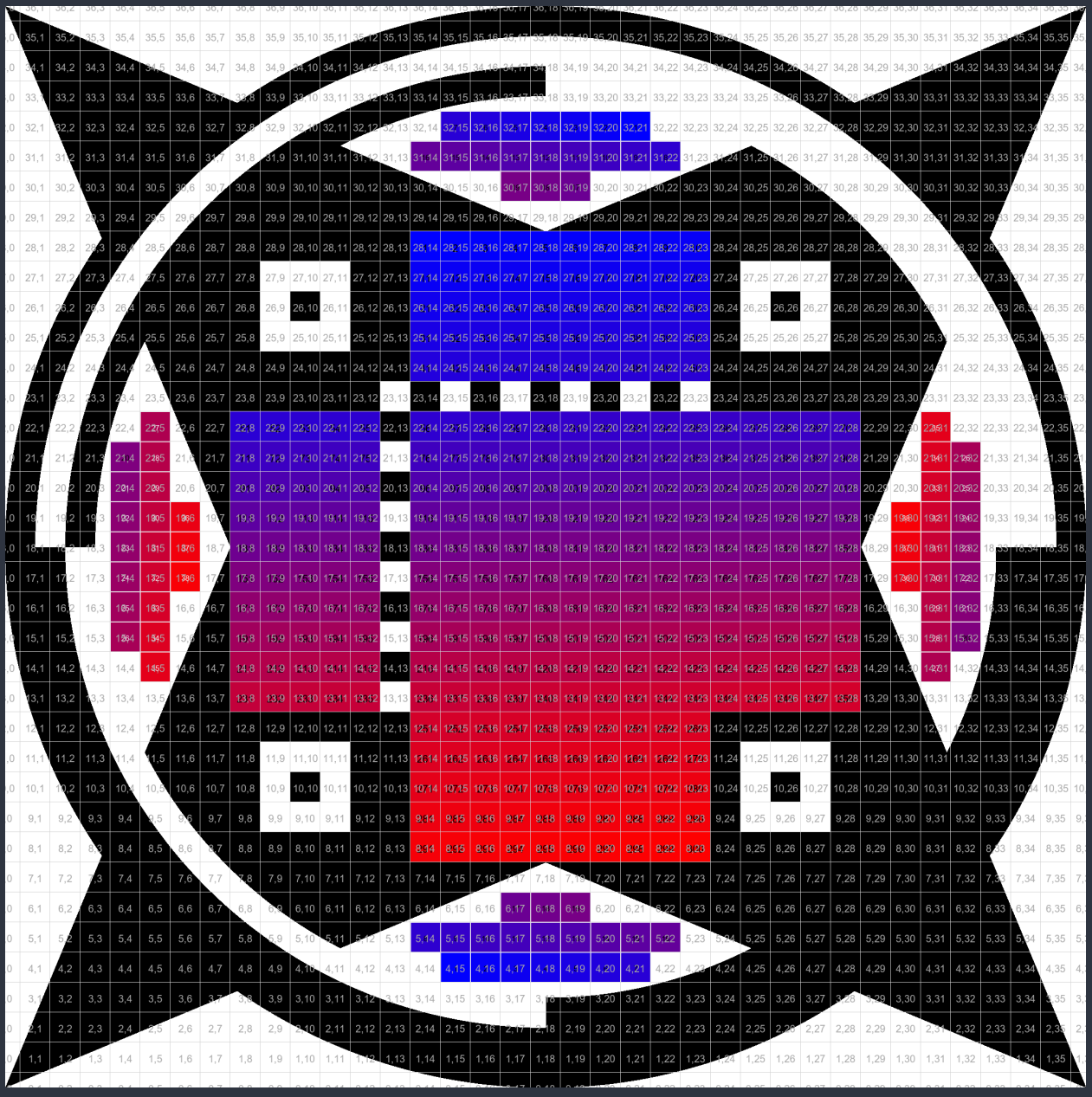

Figure 7: Fiducial Marker with Data Filling Gradient (Blue to Red)

4 CVC Adjustments for Y-Down Orientation

When working with the Computer Vision Convention (CVC) with a Y-Down orientation, several elements of the GP-Tag's design require attention to ensure correct rendering. Key differences include row indexing, quadrant definitions, rotation direction, and adjustments to the annuli and timing patterns.

4.1 Row Indexing

In the Y-Down (CVC) orientation, row indexing is inverted compared to the Y-Up (RC) orientation:

- **Row Indexing:** Rows increase from top to bottom.
  - Zero-based indexing: Row 0 is at the top, and Row 20 is at the bottom.
  - One-based indexing: Row 1 is at the top, and Row 21 is at the bottom.

4.2 Quadrant Changes

In CVC orientation, quadrants are redefined as follows while equations remain the same with updated CVC notation $\mathbf{p}^C$:

- **Q1** was the top-right quadrant in Y-Up, and is now the **bottom-right**.
- **Q2** was the top-left quadrant in Y-Up, and is now the **bottom-left**.
- **Q3** was the bottom-left quadrant in Y-Up, and is now the **top-left**.
- **Q4** was the bottom-right quadrant in Y-Up, and is now the **top-right**.

4.3 Rotation Conventions

In CVC orientation, positive rotations are defined in a clockwise direction:

- **Positive Rotation:** Moves clockwise in CVC, rather than counterclockwise as in RC (Y-Up).

4.4 Annuli Pattern Adjustments

The annuli patterns follow the new quadrant assignments in CVC with updated bit values for each quadrant.

- **Middle Annulus Bit Pattern (clockwise):**

Q1 (bottom-right):	1 (white)
Q2 (bottom-left):	1 (white)
Q3 (top-left):	0 (black)
Q4 (top-right):	0 (black)

- **Inner Annulus Bit Pattern (clockwise):**

Q1 (bottom-right):	1 (white)
Q2 (bottom-left):	0 (black)
Q3 (top-left):	1 (white)
Q4 (top-right):	0 (black)

- **Outer Annulus:** Remains a solid black boundary surrounding the tag, unaffected by orientation.

4.5 Separator and Timing Pattern Placement

The separator and timing patterns in CVC are drawn with updated row and column placements according to the Y-Down orientation:

- **Separator Pattern:** Located along Row 6 and Column 6 in CVC (instead of Row 16 in RC).
- **Timing Patterns:**
  - Horizontal timing pattern: Positioned on Row 6.
  - Vertical timing pattern: Positioned on Column 6.

4.6 Data Filling Order CVC

Data cells in the main grid and reserved areas are filled as follows in CVC:

- **Main Grid (300 cells):**
  - Filling starts from the **top row, leftmost column** and proceeds **left to right** across each row.
- **Reserved Area (76 cells), PIL Y-Down (X, Y) Coordinates:**
  - Reserved cells are arranged in **Bottom/Top** and **Left/Right** mirrored pairs for redundancy.
  - The specific coordinates for each mirrored pair and filling order are as follows:

Table 7: Reserved Area Bottom/Top Pairs (PIL Y-Down, X and Y Coordinates).
NOTE: Coordinates are in zero index convention.

Bottom Coordinate	Top Coordinate
(15,32)	(21,4)
(16,32)	(20,4)
(17,32)	(19,4)
(18,32)	(18,4)
(19,32)	(17,4)
(20,32)	(16,4)
(21,32)	(15,4)
(14,31)	(22,5)
(15,31)	(21,5)
(16,31)	(20,5)
(17,31)	(19,5)
(18,31)	(18,5)
(19,31)	(17,5)
(20,31)	(16,5)
(21,31)	(15,5)
(22,31)	(14,5)
(17,30)	(19,6)
(18,30)	(18,6)
(19,30)	(17,6)

Table 8: Reserved Area Left/Right Pairs (PIL Y-Down, X and Y Coordinates).
NOTE: Coordinates are in zero index convention.

Left Coordinate	**Right Coordinate**
(4,15)	(32,21)
(4,16)	(32,20)
(4,17)	(32,19)
(4,18)	(32,18)
(4,19)	(32,17)
(4,20)	(32,16)
(4,21)	(32,15)
(5,14)	(31,22)
(5,15)	(31,21)
(5,16)	(31,20)
(5,17)	(31,19)
(5,18)	(31,18)
(5,19)	(31,17)
(5,20)	(31,16)
(5,21)	(31,15)
(5,22)	(31,14)
(6,17)	(30,19)
(6,18)	(30,18)
(6,19)	(30,17)

5 Reference Code

Listing 1: GP-Tag Definition Overview with Grid Filling (Y-Up)

```
# GP–Tag Quick Reference (Y–Up Orientation)

# Constants (for a 360x360 pixel tag)
GRID_SIZE = 21
U = 10                  # Base unit size for 360x360 pixel tag
GRID_DIAG = 15 * U    # Diagonal of base grid

# Annuli Dimensions
r_outer = GRID_DIAG + 3 * U    # Outer radius
r_inner = GRID_DIAG            # Inner radius of inner annulus
r_inner_outer = r_inner + U     # Outer radius of inner annulus
r_middle_outer = r_inner_outer + U  # Outer radius of middle annulus

# Quadrants for Annuli Patterns (Y–Up)
quadrants = [
    {"name": "Q1 (top-right)", "middle_bit": 0, "inner_bit": 0},
    {"name": "Q2 (top-left)", "middle_bit": 0, "inner_bit": 1},
    {"name": "Q3 (bottom-left)", "middle_bit": 1, "inner_bit": 0},
    {"name": "Q4 (bottom-right)", "middle_bit": 1, "inner_bit": 1}
]

# Draw Middle and Inner Annuli Quadrants
for quadrant in quadrants:
    draw_middle_annulus_quadrant(quadrant["middle_bit"])
    draw_inner_annulus_quadrant(quadrant["inner_bit"])

# Define spike coordinates with tips and midpoints for each quadrant
spike_coords = [
    # Q1 - Top Right
    {"tip": (+r_outer, +r_outer),
     "mid1": (+GRID_DIAG / 2, 0), "mid2": (0, +GRID_DIAG / 2)},

    # Q2 - Top Left
    {"tip": (-r_outer, +r_outer),
     "mid1": (0, +GRID_DIAG / 2), "mid2": (-GRID_DIAG / 2, 0)},

    # Q3 - Bottom Left
    {"tip": (-r_outer, -r_outer),
     "mid1": (-GRID_DIAG / 2, 0), "mid2": (0, -GRID_DIAG / 2)},

    # Q4 - Bottom Right
    {"tip": (+r_outer, -r_outer),
     "mid1": (0, -GRID_DIAG / 2), "mid2": (+GRID_DIAG / 2, 0)},
]
```

```
# Fill each spike as a triangle
for spike in spike_coords:
    draw.polygon([
        spike["tip"],
        spike["mid1"],
        spike["mid2"]
    ], fill="black")  # Assuming black as the spike fill color

# Corner Markers for Alignment
corner_patterns = [
    {"x": 0, "y": 0},                       # Bottom-left
    {"x": GRID_SIZE - 5, "y": 0},           # Bottom-right
    {"x": 0, "y": GRID_SIZE - 5},           # Top-left
    {"x": GRID_SIZE - 5, "y": GRID_SIZE - 5},  # Top-right
]

# Place 5x5 finder pattern at each corner:
# [ 1 1 1 1 1 ]
# [ 1 0 0 0 1 ]
# [ 1 0 1 0 1 ]
# [ 1 0 0 0 1 ]
# [ 1 1 1 1 1 ]
for (x, y) in corner_positions:
    place_finder_pattern(x, y)

# Separator and Timing Patterns
separator_positions = {
    "row": 16,    # Horizontal black line
    "column": 16  # Vertical black line
}

# Timing pattern: alternating black/white cells
# Spans positions 6-14 on:
timing_patterns = {
    "row": 16,    # Horizontal timing pattern
    "column": 6   # Vertical timing pattern
}

# Draw timing pattern (alternating 1/0)
for pos in range(6, 15):
    draw_timing_cell(timing_patterns["row"], pos)    # Horizontal
    draw_timing_cell(pos, timing_patterns["column"]) # Vertical

# Encoding Equations for Data Values
latitude_val = int((latitude + 90) * (2**35 - 1) / 180)
longitude_val = int((longitude + 180) * (2**36 - 1) / 360)
```

```
altitude_val = int((altitude + 10000) * (2**25 - 1) / 20000)
quat_vals = [int((q + 1) * (2**16 - 1) / 2) for q in quaternion]
scale_val = int(scale * (2**16 - 1) / 3.6)

# ID Encoding
tag_id_val = int(tag_id) & 0xFFF
version_id_val = int(version_id) & 0xF

# Combine all main grid data bits
data_bits = latitude_bits + longitude_bits
          + altitude_bits + quaternion_bits + scale_bits

# Combine ID data bits
id_bits = tag_id_bits + version_id_bits

# Apply Reed-Solomon error correction 50% overhead
main_encoded = apply_reed_solomon(data_bits)
id_encoded = apply_reed_solomon(id_bits)

# Mark reserved modules (corners, separators, timing)
reserved_modules = set()
for y in range(GRID_SIZE):
    for x in range(GRID_SIZE):
        # Corner areas (5x5)
        is_corner = (x < 5 and y < 5) or                    # Top-left
                    (x < 5 and y > GRID_SIZE - 6) or        # Bottom-left
                    (x > GRID_SIZE - 6 and y < 5) or        # Top-right
                    (x > GRID_SIZE - 6 and y > GRID_SIZE - 6)  # Bottom-right

        # Separator/timing lines
        is_separator = x == 16 or y == 16

        if is_corner or is_separator:
            reserved_modules.add((x, y))

# Fill main grid with encoded data (Y-Up)
# 1 = black, 0 = white
data_index = 0
for y in range(GRID_SIZE-1, -1, -1):      # Top (high Y) to bottom (low Y)
    for x in range(GRID_SIZE):            # Left (low X) to right (high X)
        if (x, y) not in reserved_modules:
            set_grid_cell(x, y, encoded_data[data_index])  # 1=black, 0=white
            data_index += 1


# Fill reserved areas with encoded ID data (Y-Up)
# 1 = black, 0 = white
# First Bottom/Top pairs
```

```
bottom_top_pairs = [
    ((15,4), (21,32)), ((16,4), (20,32)), ((17,4), (19,32)),
    ((18,4), (18,32)), ((19,4), (17,32)), ((20,4), (16,32)),
    ((21,4), (15,32)), ((14,5), (22,31)), ((15,5), (21,31)),
    ((16,5), (20,31)), ((17,5), (19,31)), ((18,5), (18,31)),
    ((19,5), (17,31)), ((20,5), (16,31)), ((21,5), (15,31)),
    ((22,5), (14,31)), ((17,6), (19,30)), ((18,6), (18,30)),
    ((19,6), (17,30))
]

# Then Left/Right pairs
left_right_pairs = [
    ((4,21), (32,15)), ((4,20), (32,16)), ((4,19), (32,17)),
    ((4,18), (32,18)), ((4,17), (32,19)), ((4,16), (32,20)),
    ((4,15), (32,21)), ((5,22), (31,14)), ((5,21), (31,15)),
    ((5,20), (31,16)), ((5,19), (31,17)), ((5,18), (31,18)),
    ((5,17), (31,19)), ((5,16), (31,20)), ((5,15), (31,21)),
    ((5,14), (31,22)), ((6,19), (30,17)), ((6,18), (30,18)),
    ((6,17), (30,19))
]

# Fill pairs in order with ID data
bit_index = 0
# Fill Bottom/Top pairs first
for (pos1, pos2) in bottom_top_pairs:
    if bit_index < len(id_encoded):
        bit_value = id_encoded[bit_index]  # 1=black, 0=white
        set_grid_cell(pos1[0], pos1[1], bit_value)
        set_grid_cell(pos2[0], pos2[1], bit_value)
        bit_index += 1

# Then fill Left/Right pairs
for (pos1, pos2) in left_right_pairs:
    if bit_index < len(id_encoded):
        bit_value = id_encoded[bit_index]  # 1=black, 0=white
        set_grid_cell(pos1[0], pos1[1], bit_value)
        set_grid_cell(pos2[0], pos2[1], bit_value)
        bit_index += 1
```

Listing 2: GP-Tag Scale Calculations

```
# ------------------------------
# Bonus: Scale Calculations
# ------------------------------

# Method 1: Calculate scale from U and DPI
scale_from_U_DPI(printer_dpi, base_unit_U):
    Input: printer_dpi (e.g. 600), base_unit_U (e.g. 10)
    Output: scale (cells/mm)

    dots_per_mm = printer_dpi / 25.4      # Inches to mm
    mm_per_cell = base_unit_U / dots_per_mm
    scale = 1 / mm_per_cell               # Cells per mm
    return scale

# Method 2: Calculate U and scale from size
U_scale_from_size(printer_dpi, size_mm):
    Input: printer_dpi (e.g. 600), size_mm (e.g. 100)
    Output: base_unit_U, scale (cells/mm)

    dots_per_mm = printer_dpi / 25.4      # Inches to mm
    total_cells = 36                      # 36x36 tag
    U = round((size_mm * dots_per_mm) / total_cells)
    scale = total_cells / size_mm         # Cells per mm
    return U, scale
```